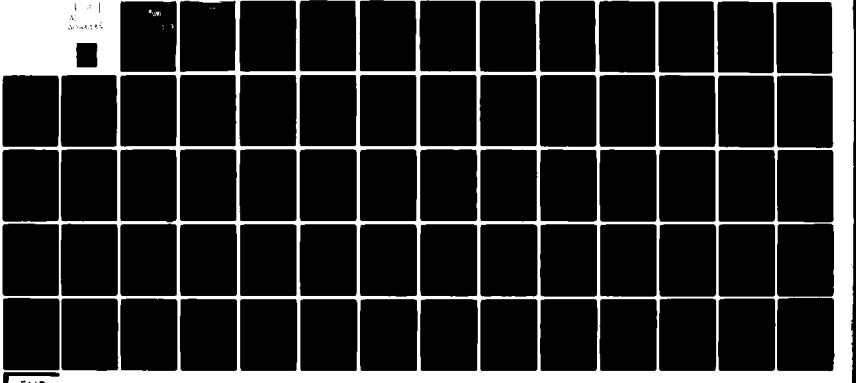
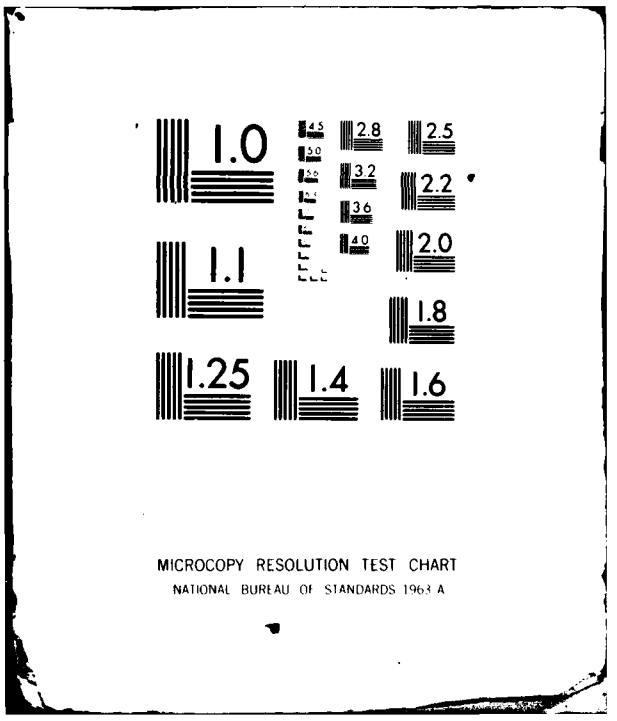
AD-A093 439 ROYAL AIRCRAFT ESTABLISHMENT FARNBOROUGH (ENGLAND) F/G 8/2
A CARTOGRAPHIC COMPUTER DATA FORMAT AND ASSOCIATED PROGRAMS.(U)
MAR 80 A H BENNY
UNCLASSIFIED RAE-TR-80037 DRIC-BR-74346 NL



END
DATE FILMED
1-81
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

UNLIMITED

TR 80037

AD A093439

LEVEL II

ROYAL AIRCRAFT ESTABLISHMENT

* RAE-TR-8

(9) Technical Report 80037

(11) March 80

DTIC ELECTE JAN 9 1981 S D E

(6) A CARTOGRAPHIC COMPUTER DATA FORMAT AND ASSOCIATED PROGRAMS

by

(12) 68

(10) A.H. Benny

(13) DRIC

*

(14) BR [illegible]

Procurement Executive, Ministry of Defence
Farnborough, Hants

DDC FILE COPY

320450

UNLIMITED 80 12 [illegible]

Report Documentation Page

Control Data

UNLIMITED

Instructions for completion appear overleaf.

As far as possible this page should contain only unclassified information. If it is necessary to enter classified information, the box above must be marked to indicate the classification, e.g. Restricted, Confidential or Secret.

1. DRIC Reference (to be added by DRIC)	2. Originator's Reference RAE TR 80037	3. [illegible] N/A	4. [illegible]
5. DRIC Code for Originator 7673000W	6. Originator (Corporate Author) Name and Location Royal Aircraft Establishment, [illegible]		
5a. Sponsoring Agency's Code N/A	6a. Sponsoring Agency (Contract Authority) Name and Location N/A		

7. Title

A cartographic computer data format and associated programs

7a. (For Translations) Title in Foreign Language

7b. (For Conference Papers) Title, Place and Date of Conference

8. Author 1. Surname, Initials Benny, A.H.	9a. Author 2	9b. Authors 3, 4 ...	10. Date [illegible]
11. Contract Number N/A	12. Period N/A	13. Project	14. Other Reference Nos. [illegible]

15. Distribution statement

(a) Controlled by – Head of Space Department, RAE

(b) Special limitations (if any) –

16. Descriptors (Keywords) (Descriptors marked * are selected from TEST)

Data format. Cartography*.

17. Abstract

This document explains the need for a new computer data format for cartographic data previously available, for the storage of cartographic data.

A suitable format is described, together with a number of computer programs which have been written to enable data files in this format to be created, manipulated and displayed in various ways.

UDC 526.8 : 681.3.07 : 519.688 : 531.7.084.2 : 629.19

ROYAL AIRCRAFT ESTABLISHMENT

Technical Report 80037

Received for printing 10 March 1980

A CARTOGRAPHIC COMPUTER DATA FORMAT AND ASSOCIATED PROGRAMS

by

A. H. Benny

SUMMARY

This document explains the need for a new computer data format, not previously available, for the storage of cartographic information.

A suitable format is described, together with a set of computer programs which have been written to enable data files in this format to be created, manipulated and displayed in various ways. → p. 3

Departmental Reference: Space 578

Copyright
©
Controller HMSO London
1980

LIST OF CONTENTS

Accession For
NTIS GRA&I X
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

1 INTRODUCTION

Computer data can be stored in a variety of ways and each method is likely to have its own advantages and limitations. This document is concerned with the storage of data representing some form of 'picture' of the Earth's surface, as viewed almost vertically downwards from one of the 'Landsat' series of orbiting satellites, using its multispectral scanner (MSS).

The Landsat MSS is described in detail in Ref 1. A simplified description will suffice to convey the information needed here. As the satellite moves above the Earth, its surface is scanned by means of an oscillating mirror, which deflects light, via an optical system onto a sensor system. The mirror oscillation is perpendicular to the path of the satellite, so that a raster-scan is made, and during each west to east scan the output of the sensor is digitized at regular intervals of time. Simplifying somewhat, the result is that a swath of land is imaged, to form a raster-scan of radiometric values with a constant number, L of values per scan. If the process is stopped after M scans have taken place, the result is a series of V data values, where $V = L \times M$.

1.1 The 'image' data format

It is convenient to store these V values in the sequence in which they were obtained, to form a data file. Provided that the number of values per scan is known it is then possible to reconstruct the picture by a suitable raster-scan display, with each picture element (pixel) having a brightness corresponding to its measured value. This illustrates a simple and useful form of data storage - a sequence of values, which is known to represent a raster-scan. It should be noted, however, that the number L of values per scan must be known (otherwise the picture cannot be reconstructed) and preferably M also, so that the size of the file (and picture) is known in advance. It is therefore usual to have some form of information - a 'header record' - before the main body of the data, so that the latter can be properly handled.

A format as described is now in use in Space Department of RAE. In detail each data file consists of a continuous sequence of 8-bit values. The first 88 such values constitute the header information, and include a value L for the number of pixels per scan line. (In practice the number M of scan lines is also included.) The following data consist of M sets of L values, the 'brightness' of the pixels of the scene. Such data files are conventionally referred to as image files or images, and the programs which handle them are conventionally

prefixed by the code IM . Thus, for example, program IM.LOOK is used to 'look at' an image file in a certain way.

The data storage method described above has several advantages; in particular it is simple to use and economical of storage space. However, it also has some limitations, such as:

(a) Each pixel is located, by implication at a specific place, an integer number of pixels along and down the picture. Fractional locations cannot be referred to.

(b) It is not easily possible to establish any relationships between pixels, beyond their obvious sequential nature. For example, pixels in adjacent scan lines are not obviously related. Any feature on the original scene which extends over more than one scan line is not recognisable as such from the data, (though it may be readily observed by human beings from the resulting picture display).

1.2 The 'map' data format

Considerable work is being done worldwide on the interpretation of Landsat data. Sometimes the resulting information can be conveniently stored in an image file. Some recent interpretation work done at RAE[2] has been concerned with the extraction of cartographic features, *eg* coastlines, and these are not suitable for storage in image files, in particular for the two reasons mentioned in (a) and (b) above. Consequently, it was necessary to establish a more suitable data storage format.

Briefly, the new data format, referred to as a 'map' format, consists of a header record, followed by a number of 'features', each feature consisting principally of either one coordinate value (to represent a single location - a 'point'), or a string of coordinates (to represent a continuing 'line'). Such a map data format is capable of storing much of the diagrammatic information extracted from a Landsat image.

This document describes the map data format in detail, together with a description of a suite of computer programs and subroutines which has been written to handle such data files. Programs which handle map files are conventionally prefixed by the code MAP., *eg* the program MAP.LIST is capable of listing the coordinates of a map file.

The expression 'line' or 'linear feature' of a scene is here used to refer to a sequential set of coordinates (x, y pairs). If the commencing and final

037

coordinates of a line have the same coordinate values, in both x and y , then the line is closed, *ie* it forms a loop. Because the connections between each coordinate pair can only be considered as straight lines, a loop is in fact a polygon. If the individual coordinates are sufficiently close together, the linear feature, when displayed on some suitable device, such as an X-Y plotter or a graphics visual display unit, appears to the eye as an acceptable representation of a terrestrial feature, that is to say, the display resembles a map etc.

2 THE MAP DATA FORMAT

A map data file consists of a header record followed by the body of the data, in sequence. For convenience, and with the objective of maximising the reading and writing speed, the file is divided into 'blocks', each of 1024 words, since that is the number of data words per storage block of the disc unit of the host computer (Prime) on which the file is to be stored. Should files be written to other machines, the block size should be changed suitably, and provision has been made for such a change to be done with the minimum of work.

The header record is therefore 1024 words in length, and the body of data is divided into similar sized blocks, although individual data features may be larger or smaller than 1024 words. The Prime computer uses 16-bit words, and the map format has been designed accordingly.

2.1 The header record

The header record has 1024 words available to it, and this is considerably in excess of the amount normally needed. This space is allocated as shown in Table 1, the details of which are as follows.

2.1.1 'Permanent' header description

Programs which create a map file normally call the subroutine PERMHD which requests from the user the permanent header details, these being the general project name IPROJ , up to 12 characters, plus the description of the project, IDESCR , up to 32 characters. These details are not changed by subsequent processing of the file, but are permanently available for reference.

2.1.2 Map projection details

Since the map format is capable of holding (and does on occasion hold) data which is the computer description of a map, it is useful to have the facility to record some information relating to the cartographic projection used.

037

Some or all of the following information may be recorded:

(a) IPRJC1: a code number which indicates the type of projection being used. Codes so far allocated are:

IPRJC1	Projection
1	British National Grid
2	Universal Transverse Mercator
99	Image file units.

The code 99 is not strictly a map projection, since it is not cartographically defined. It is related to the movement of the Landsat satellite. Methods exist for transforming 'maps' in this pseudo-projection into accepted cartographic projections however, so it is useful to label a file with the projection code 99.

(b) IPRJC2: a projection subcode. This number may be used to qualify the main code. The only meaning so far allocated is for the case when IPRJC1 = 2 , *ie* UTM. In such cases it is necessary to specify the zone, (1 to 60) and this value is held in IPRJC2 .

(c) SCALE: this real number is the scale of the map. For example, for a one-inch to the mile map, SCALE would have the value 63360.0.

(d) XSCALE, YSCALE: not at present in use. If it should be necessary to adjust the size of a map to fit its known corners, the X and Y adjustment factors would be stored in these locations. Normally their values would be close to unity.

(e) SCLLNG, SCLLAT: these numbers are available for holding longitude and latitude values, usually stored as degrees. For example, in a UTM map, SCLLNG could contain the value of the central meridian of the zone; for a Mercator map, SCLLAT could contain the value of the latitude at which the SCALE is accurate.

(f) RADEA, RADEB: the Earth's semi-major and semi-minor axes in metres. These values are not at present in use, but are needed if map projections are to be considered in detail, or if maps are to be transformed from one projection to another.

2.1.3 Map file statistics

This set of values holds statistics about the map file, which are useful for various purposes.

(a) NLIN: the number of line items in the file.

(b) NPT: the number of point items in the file. Since a map file should always contain four corner points or 'fiducials', NPT should never be less than 4.

(c) NSTEPS, NSTEPZ: the total number of line 'steps' in a file is held in these two values. Any one line may have N coordinate values, including its start and finish, and thus has N-1 'steps'. The number of steps for each line is totalled, and the value stored. Since this value may be too large to hold in the 16-bit integer available, it is stored as two integers, modulus 10000. That is to say, if NTOT is the total number of steps, then NTOT = (NSTEPZ × 10000) + NSTEPS .

It would have been possible to use a REAL word or an INTEGER*4 to store this information. The chosen method allows larger numbers to be handled, and more conveniently, than a REAL . INTEGER*4 is not available as a standard FORTRAN facility (though it is becoming commonly available).

(d) MNFC: each item of the map file can carry up to 17 'feature codes', which give information about the type of feature the item represents. MNFC records the largest number of such codes carried by any item in the file.

(e) NDIFFC: a file may contain many items, and each item may carry several codes. The total number of different feature codes in a file is recorded by NDIFFC . Usually NDIFFC is larger than MNFC .

2.1.4 Corner points

A map is a representation, on say a sheet of paper, of a portion of the surface of the Earth. Two sets of corner points exist therefore: the corners of the map and the corresponding locations on the Earth's surface. The eight real words of array CNR(2,4) hold the x and y coordinates of the four corners of the map, reading from the south west corner in a clockwise direction.

```
CNR(1,1)    x  coordinate of SW corner
CNR(2,1)    y  coordinate of SW corner
CNR(1,2)    x  coordinate of NW corner
   :
   :
CNR(2,4)    y  coordinate of SE corner.
```

As a convention, the SW corner is taken as the origin of the map, so its coordinates are 0, 0 . Also, the map is defined as a rectangle. With these restraints, considerable redundancy exists, and in fact all of the required information may be derived from the NE corner.

The CNR values are stored as millimetres. Arbitrarily, a maximum map size of 2000 mm square may be defined.

Corresponding to the four corners on the map are four geographical locations on the Earth's surface. These locations are stored in the array GCNR(2,4), and they correspond with the CNR values, *ie* GCNR(a,b) corresponds with CNR(a,b) .

If the map is in British National Grid, the GCNR values are in kilometres on that grid. For other map projections GCNR values are held as the relevant longitude and latitude, expressed in radians. For IPRJC1 = 99, *ie* the pseudo-projection based on image file units, GCNRs have no defined meaning at present.

2.1.5 Program logging information

The header record has a logging facility to allow a record to be made of all processing done to a map file. Some programs are available to modify existing files, and it is useful to keep a log of what has been done. All such programs, together with the program which initially created the file, use a subroutine UPDATE , which keeps a log in the header record.

Subroutine UPDATE gathers information, some from the user, but mostly from the computer itself and from the program in use, and places this information into a 40 word portion of an array (only 29 of these words are at present in use). There is space in the header record for 15 such sets of 40 words. After a 40 word set has been assembled, UPDATE moves all the preceding sets one place 'down', and then places the latest set at the top of the available space. Thus, if 15 sets of log data had been in use prior to this operation, the oldest one would have been lost, as the others moved down. In practice, it is not expected that there will be as many as 15 UPDATES on most data files, with the suite of programs at present available.

Within each 40 word set of data, the information recorded is:

(a) the name of the current program - up to 8 characters;

(b) the version number of the current program. From time to time program changes may be made sufficient to justify a new version number;

(c) the version number of the data file. Each time UPDATE operates, it increments this value by one;

(d) the current date and time. This information is extracted by the computer system from its own records. The method of storage of these values is seen by a study of the listing of the subroutine UPDATE ;

(e) the user's code name. As with (d), this is provided by the computer system;

(f) details of the current program operation. Up to 32 characters of description may be supplied by the user, in response to a request for "details of this run".

2.2 Data records

The cartographic data in a map file follows the header record, and is stored in similar sized data records. The data consists of a sequence of map 'features', these being of three types; corner points, points, and lines. The two types of point contain a relatively small amount of information and each is stored as one 'item', but lines can contain an indefinitely large amount of information, so they are broken down into a number of items, one for each coordinate pair of the line, the first and last coordinate pairs being in rather different types of item from each other and from the 'internal' ones.

Map data is thus broken down into items, which may be from 2 to 21 real words in length, though the average item size is probably under three words for a typical map. These items are stored within data records of length 512 real words (1024 integers), and each type of item may carry an indicator word (denoted by MK) according to its type (see Table 2).

Table 2

Meaning of MK numbers

MK	Meaning
0	'Internal' line coordinate pair
1	Last coordinate pair of a line
2	Not used
3	Start of a line, including first coordinate pair
4	Map point
5	Corner point or 'fiducial'
6	End of data indicator
7	Fill word

Since items are stored without being broken, there is often one or more words unused at the end of a data record. In such cases the first or only empty word has a 'fill' word (MK = 7) placed in it, to distinguish it from a zero.

037

The condition MK = 7 is written and read by subroutines NSTORE and NFETCH respectively. It is not used in the calling program, nor need its existence be considered by the writer of calling programs, for it is to him an illegal condition. It is included in Table 2 for completeness only.

When the last map item has been written, a MK = 6 item is placed after it, to indicate that the end of the data has been reached.

In addition to the coordinate pairs, where appropriate, the item may also carry some information about the feature, and this is done for start-of-line and map point items. Both such features may carry up to 17 feature codes, so the item includes NFC , the number of feature codes for this item, followed by NFC real words containing the relevant feature codes. Additionally the start-of-line item has provision to record the number of steps in that line (*ie* the number of coordinate pairs less one). The number of steps is broken down into two integer values, NPTS and NPTZ such that the total number of steps is NPTZ × 10000 + NPTS . In practice, lines vary in length from 1 to about 50000 steps, so NPTZ seldom exceeds say 5. If the number of steps is not known, zero should be inserted in NPTS and NPTZ .

In the case of 'internal' line coordinate pairs, the inclusion of the MK word would lead to a 50% increase in item size, and since a map often consists mainly of such items, almost a 50% increase of data file size. Thus, for internal points, the MK word is not stored. However, it is necessary to supply MK = 0 when internal coordinates are to be written by subroutine NSTORE , and after the retrieval of such data by NFETCH MK = 0 is returned.

The constitution of the various types of item is shown in Table 3 (section 2.2.1).

Since the data storage method is almost entirely encompassed by the subroutine NSTORE (and correspondingly the recovery of the data by NFETCH) the former will now be described in some detail.

2.2.1 Subroutine NSTORE

The file data must be stored in such a manner that it can be recovered unambiguously, and this is done by subroutine NSTORE . This subroutine communicates with the other portions of the program via two COMMON blocks, the first of which, COORDS , must be supplied with all the information necessary for the storage operation, and the second, OUTCOM , contains the output buffer and its pointer together with other details necessary for the output operation of writing to the storage medium.

037

The COMMON block COORDS contains the values:

(a) MK: an indicator of the type of item etc being stored.

(b) Two real values, the x and y coordinates of the item.

(c) Two integers, representing the number of steps in the line. These are only used if the item is the start of a line.

(d) One integer NFC representing the number of feature codes for the item.

(e) Up to 17 real values, the actual feature codes.

(d) and (e) are only supplied if the item is a map point or the start of a line. The number of codes supplied in (e) should be consistent with (d), as NSTORE stores NFC values, irrespective of whether these have been deliberately supplied or not.

The COMMON block OUTCOM contains in particular a buffer pointer which points to the next free word in the output buffer.

The items placed into the buffer by NSTORE are made up as shown in Table 3.

Table 3

Types of item stored

MK	R	X	Y	NFC	Feature codes	Item size (words)
0		✓	✓			2
1	✓	✓	✓			3
2						Illegal
3	✓	✓	✓	✓	NFC codes	4 + NFC
4	✓	✓	✓	✓	NFC codes	4 + NFC
5	✓	✓	✓			3
6	✓					1
7	(✓)					(1)

The only legitimate MK values which may be supplied to NSTORE are 0, 1, 3 to 6, and all others cause an error message to be output. NFETCH recognizes MK = 7 as a fill word.

Subroutine NSTORE starts by using MK , and if necessary NFC , and calculates the size of the item, as shown in the last column of Table 3. It

then uses the buffer pointer and buffer size to determine whether sufficient space remains to write the item to the buffer. If insufficient space remains, it 'closes' the buffer and writes it to the output device. Closing the buffer consists of examining it to see if any empty space remains, and if so, an R value corresponding to MK = 7 is placed in the first or only empty space. R is a real number, made up of the addition of MK + 1048576.0 . After writing the buffer to the output device, the buffer is cleared and its pointer reset.

In the event that sufficient space remains in the buffer, or a buffer has just been written and cleared, the item, as described in Table 2, is then loaded into the buffer. Much of this operation is straightforward, but two points need elucidation.

(a) It is necessary for NSTORE to write R (MK + RMARK) rather than merely MK , to avoid ambiguity when the data is later recovered by NFETCH . If only MK were used, the possibility of confusion would occur when line-points were being recovered. At any time, it will not be known whether the next item is another line-point (in which case the next word will have the value x) or the end of the buffer (MK = 7) or the end of the line (MK = 1). Since 1.0 and 7.0 are legitimate values for x , confusion could arise. Hence MK has RMARK added to it, RMARK being large enough to exceed any allowed value of x yet small enough for the addition of MK (1 to 7) to allow subsequent recovery of MK by NFETCH.

The user of subroutines NFETCH and NSTORE need not be concerned with the details of the construction of R since it is handled internally by these two subroutines and no others.

(b) The values of the three integers NFC , NPTS and NPTZ are combined into one real word. This is done for reasons of economy; as NFC cannot exceed 17, and NPTZ is in practice seldom greater than say, 5 to 10, these two small integers can conveniently be packed into one 16-bit word, which in fact allows NPTZ values up to 1023.

3 DESCRIPTION OF PROGRAMS FOR CARTOGRAPHIC DATA FILES

3.1 Brief description of the programs

Map files must first be created; they may then be modified in various ways; they can be transformed into other forms of file; and they can be examined in several ways. Programs have been written to perform a number of these operations. This section describes the programs briefly; a fuller description and program listing appears later.

037

3.1.1 Programs to create map files

(a) MAP.FAB This program allows the user to create a map file manually, by keying all the details into a VDU . Its main use is for such purposes as to form an 'empty' map - *ie* merely four corner points, or to create a grid of horizontal and vertical lines, although one could, laboriously, key in a whole map if the data were available in coordinate form.

(b) MAP.IN This program creates a map file from an external 'exchange file' produced elsewhere and stored on magnetic tape.

(c) IM.CONTOUR This program is named in the 'IM.' series, because it uses an image file as its starting or input material, extracts cartographic lines from it, and forms a map file. The program is to be described in a separate document, and has been described in a general way in Ref 2, so will not be detailed in this document. However, it is at present one of the main sources of map files.

3.1.2 Programs to alter or transform map files

(a) MAP.HDRED It is sometimes necessary to inspect, alter or add to the information contained in the header record, *ie* edit the header, and this may be done by use of MAP.HDRED .

(b) MAP.TRANS Map files created by IM.CONTOUR are in the pseudo-projection of image file units. With the aid of a suitable transformation matrix, MAP.TRANS can be used to transform such a map file to one in a known map projection, such as British National Grid.

(c) MAP.OUT This program uses a map file as input and creates an 'exchange file' on magnetic tape, which may then be despatched elsewhere, or could of course be re-read by MAP.IN .

3.1.3 Programs to examine map files

(a) MAP.LIST This program displays to the user, on a VDU , some details of the header, and the data, item by item, all in alphanumeric form. To avoid the display of large quantities of data, the user may make some degree of selection of what is to be displayed.

(b) MAP.TEK A map file may be graphically displayed on a Tektronix 4014 device by means of program MAP.TEK .

(c) MAP.HPLOT This program allows a map file to be plotted on a Hewlett-Packard X-Y plotter. Both MAP.HPLOT and MAP.TEK allow the user some degree of control over the material displayed and the size of the display.

3.1.4 List of programs

In alphabetical order, the programs above are:

Name	Section	Description
IM.CONTOUR	3.1.1c	Ref 3
MAP.FAB	3.1.1a	3.2.1
MAP.HDRED	3.1.2a	3.2.2
MAP.HPLOT	3.1.3c	3.2.3
MAP.IN	3.1.1b	3.2.4
MAP.LIST	3.1.3a	3.2.5
MAP.OUT	3.1.2c	3.2.6
MAP.TEK	3.1.3b	3.2.7
MAP.TRANS	3.1.2b	3.2.8

The above list of programs is not intended to be exhaustive, but includes all of those so far written. There may be a need for including additional features in these programs or writing of new ones for other tasks. For example, it may soon be useful to be able to create map files by digitization from a map or other diagram using the graphics tablet at present available.

3.2 Detailed description of the programs

With the exception of IM.CONTOUR , which is to be described elsewhere, the programs are next described in more detail, the aim being to aid potential users in understanding how to operate them. The program listings are provided at the end of this document, and these may be referred to whilst studying the following.

The programs make use of a number of subroutines which have been placed into a 'library', and are described in section 4.

3.2.1 Program MAP.FAB

Program MAP.FAB allows the user to create a map file, by supplying data from a keyboard.

From the user's viewpoint, its operation is as follows:

* The program name and writing date is output.
* An output filename (*ie* the file to be created) is requested.
* Permanent header detail is requested.
* Description of the current run is requested. The header record is then written to the storage medium. If further header detail is to be supplied, this is done by subsequent use of program MAP.HDRED .
* A 'MK' number is requested. Subsequent action depends on the reply given.

  (a) MK = 6 causes the output file to be closed and the program terminates.

(b) Illegal MK numbers invoke an appropriate message, and a further request for a MK number.

(c) Legal MK numbers are followed by requests for the data needed to create an item appropriate to that MK number. When the necessary answers have been provided the item is written by NSTORE and a request for another MK number is made.

It is thus seen that this program creates a map file, with the header record in place and partially completed, and followed by data items, each checked for internal correctness, and terminated by a MK = 6 item. There is however no check that the sequence of the data as a whole is correct: for example, fiducials may be written at any time, and a line may be ended before it has been started. Checks could be made for such errors, but it is thought desirable to allow the user to include deliberate mistakes, to test other programs.

3.2.2 Program MAP.HDRED

The header record is seen (Table 1) to consist of a number of values, some integer, some real and some characters, in the first 132 (integer) words. Words 201 to 800 contain the history of the file.

The program MAP.HDRED allows the user to examine, and if desired change, any of the information contained in words 1 to 132. If any change is made, the history is updated to include this fact. The name of the file is not changed.

The operation of the program is as follows:

* The program announces its name and version date.

* The name of the data file is requested.

* The question 'HEADER?' is asked. If answered YES the project and description are listed, together with the history of the file, as stored in header words 201 onwards.

* Next an 'ITEM' is requested. Thirty-six different answers are acceptable as shown in Table 4, otherwise the request will be made again, after a message stating that the item is not recognized. If the user replies HELP , the 36 recognisable options will be listed for inspection. If END is typed, or merely a carriage-return, the program closes down, updating the header and writing it back into the data file if necessary (if any change has been made). If any of the recognised names are typed, the value of that quantity will be displayed, and the question posed: CHANGE IT?

If NO is replied, no action is taken, and the program asks for another ITEM . If YES , then NEW VALUE is requested, and the value provided is written into the header buffer. No check is made by the program concerning the validity of the value provided, so the user must take care to ensure that the required value is entered correctly. If there is doubt, or a mistake is known to have been made, it is possible to re-examine and alter the altered value.

* When the user has replied END to the query 'ITEM' the program checks whether any change has been made (a change is said to have been made when the reply YES is given to the query CHANGE IT? even if the same value is given as existed before) and if so, it updates the header and writes it back to the storage medium. The file is then closed.

* Finally the program requests MORE FILES? to enable the user to edit the header of another file should he so wish.

3.2.3 Program MAP.HPLOT

Program MAP.HPLOT is used to plot a map file on a Hewlett-Packard X-Y plotter. The interface software to drive that plotter has been obtained from elsewhere and is therefore not described here. The various subroutine calls, *eg* HPINIT, HPMOVE , and so forth, are noted in the program listings with comment to indicate their functions.

User operation is described below. Before running the program, the user must assign the plotter to the current VDU , by the instruction: AS AMLC 15

* Program name and date is output to the VDU .

* The pen number (1 to 4) is requested. The user may insert pens with coloured ink into some or all of the four positions available.

* The cross size N is requested. Corner and other points are drawn as crosses, the arm length of which will be N units, each one-eighth of a millimetre. Hence the reply 10 would produce crosses of arm length 1¼ mm, *ie* 2½ mm from tip to tip.

* A request is made for the name of the input map file.

* An enquiry is made; 'SPECIFY CORNERS?' If the reply is YES , then requests for x and y coordinates of the south west and north east corners (*ie* lower left, upper right) will follow, and the plot will be adjusted so that the map, based on the corners in the header (CNR), fits the numbers provided.

037

* If the map file has IPRJC1 = 99 , *ie* it is in image file units, the program moves to the enquiry FRAME? below. Otherwise, if the above question were answered NO , the next request would be 'SPECIFY SCALE?'. If this is answered YES the program will refer to the SCALE in the header and if it finds no value there (actually SCALE less than 10) a suitable message will be output. Provided that a suitable SCALE is available, the program will request an 'OUTPUT SCALE' . A check will then be made to ensure that the map, when drawn to this scale will fit the size of the plotter and if unsuitable, a fresh value will be requested. The scale should be provided as a real number, as shown in section 2.1.2(c).

* If the answer to the question 'SPECIFY SCALE?' had been NO , then the program would calculate the necessary factor, such that the output map would just fit the height or width of the plotter, whichever is the limiting factor.

  Thus the user has a choice of three types of map size, or two if there is no value for SCALE in the header.

* An enquiry 'FRAME?' is made. If the response is YES , the plotter will draw a frame round the map, *ie* a line joining the four corners, in sequence and returning to the starting corner.

  Following these questions, the X-Y plotter will then draw the map, under computer control, using the data from the input file. When the plot is complete;

* the query 'ANY MORE FILES?' is made. If YES , a return is made to the first question; if NO , the program stops.

3.2.4 Program MAP.IN

For purposes of data exchange or transfer between users in different establishments, a data 'exchange format' has been defined, and this is described in Appendix A. One or more maps may be contained on such a magnetic tape. The program MAP.OUT converts data files into exchange format data on magnetic tape, and MAP.IN correspondingly creates one or more map format data files from an exchange format magnetic tape.

Before the program MAP.IN is run, the user must first allocate or 'assign' a magnetic tape drive unit for his use, and load the magnetic tape containing the exchange format data onto that drive unit. The program may then be run:

* The program announces its name and asks for the magtape drive unit number. The user must supply the unit number previously allocated.

037

* The first ten magnetic tape records are read by the computer, interpreted as alphanumeric characters and displayed on the VDU . These ten records usually give some descriptive details of the map to follow.

* The usual filename, permanent header data and run description, are requested. The user is able to provide suitable replies, often based on the ten records just supplied.
  Following these questions, the computer then reads the magnetic tape, converting the data into a suitable form for the map file being created on the storage medium (usually a disc), until the first (or only) map has been completed.
  If no more files exist on the magnetic tape (and this is recognized by the presence of a MK = 30 or an end-of-file) then the tape is rewound and the program stops. If, however, another file (or files) is present, the ten character records are displayed on the VDU , and the program continues from that location.

Thus, one magnetic tape in exchange format may result in the formation of several data files. Each of these data files will have only a small amount of header record detail, consisting of the file descriptive character words, the update information, and the map corners (CNRs) derived from the corner points. If it is desired to include further header information, *eg* the geographical corners, or the scale, which may have been provided by the ten lines of text message, then MAP.HDRED may be used to enter this.

The exchange format allows one or more records per magtape block. Both MAP.IN and MAP.OUT have been written for the case of one record per block, which has been suitable for the data transfers so far performed. Should it be necessary to use multi-record blocks, some program modification would be necessary.

3.2.5 Program MAP.LIST

The contents of a map data file may be inspected in the form of characters and numbers on a VDU by means of program MAP.LIST . When run:

* The program announces its name and date.
* A map filename is requested and the file opened.
* The question HEADER? is posed. If the user replies NO , the program proceeds to the data, but if YES , a number of header details is displayed on the VDU . Only the more commonly inspected details are shown, but the less frequently used details may be seen with the aid of MAP.HDRED .

* The question DATA? is asked. If the reply is NO , the program moves on to a later question, but if YES the question INTERIOR LINE POINTS? follows. If this is answered YES , then all of the map data items are displayed, in sequence, on the VDU . The reply NO would cause the interior line point items (MK = 0) to be omitted, which is often convenient.

* Following the data, the program asks MORE FILES? If YES , a return is made to the first question in the program, but if NO the program stops.

3.2.6 Program MAP.OUT

Program MAP.OUT performs the reverse operation of that done by MAP.IN , and reference may usefully be made to the description of the latter program, in section 3.2.4.

MAP.OUT is used to form a magnetic tape, in exchange format, containing one or more map files.

Before MAP.OUT is run, the user must assign a magnetic tape drive unit for his use, and load onto that unit the tape on which the exchange data is to be written. The program may then be run.

* The program provides its name (on the VDU) and requests the magtape drive unit number.

* The name of the input map data file is requested, and the named file then opened.

* Ten header lines are requested. The user must supply ten lines of text, from 0 to 120 characters per line. Since the exchange format does not contain headers, it is usual to write out some of the header information as text.

  The data is then converted to exchange format and written onto the magtape. When the process is completed the program asks ANY MORE FILES? If YES , the name of the next file is required, and the above steps continue. If NO , MK = 30 and end-of-file markers are placed on the tape, which is then rewound, and the program stops.

3.2.7 Program MAP.TEK

Program MAP.TEK is used (in a manner analogous to MAP.HPLOT) to display a map file on a Tektronix 4014 graphics display unit, with hard copy capability. As with HPLOT , the software which drives the display has been provided from elsewhere.

037

For operation, the user must first assign the display to his use by making the instruction:

AS AMLC 14 TRANHS 413

with corresponding un-assignment:

UN AMLC 14

after the program use has been completed. The assignment instruction may be changed at some later date, *eg* the AMLC line may no longer be number 14.

When the program is run:

* it provides the program name and date on the VDU,
* it asks if the user wishes to 'flash' the screen, *ie* clear it of its present contents,
* it requests CROSS SIZE , N . All map points are drawn as crosses, having four arms, each of length N units. These units are related to the size of the display screen but are approximately 0.1 mm. Hence a reply of N = 10 would provide crosses with arms of length about 1 mm, *ie* 2 mm from tip to tip.
* the filename is requested from the user, and when supplied the map file is opened,
* SPECIFY CORNERS? is asked. If the reply is YES the user is asked for the x and y coordinates of the south west and north east corners. Values should be in screen units, as mentioned above, and normally these would be provided in the range 0 to 5000. It is possible to use specified corners to fit a map file to a displayed image file. If the user replies NO the program refers to the corner points (CNR) in the header record of the file, and calculates a scaling factor such that the map is drawn as large as possible consistent with (a) scales in the x and y direction are the same and (b) the map fits entirely into the screen.
* The program asks FRAME? If YES , four straight lines are drawn, clockwise from corner to corner, creating a frame round the map.
The map file is then drawn on the screen, feature by feature, until it is complete. The file is then closed.
* ANY MORE FILES? is requested. YES causes a return to the initial 'FLASH THE SCREEN?' question, ready for another map file to be displayed, whilst NO causes the program to stop.

If the corner (CNR) values in the header record of a file are either zero (perhaps none has been provided) or corrupt, an error message will be displayed and that file closed. In these circumstances it is possible to use program MAP.HDRED to insert or correct the CNR values. The CNR values can be selected by examination of the fiducial points (the first four data features of the file) using program MAP.LIST .

3.2.8 Program MAP.TRANS

Map data files produced from image files by means of programs such as IM.CONTOUR are not in a cartographically recognised map projection, but are in a coordinate system related to the initial image. This system has been referred to as Space Oblique Mercator[4]. The program MAP.TRANS may be used, together with a suitable transformation file, to convert the map file to a coordinate system such as British National Grid.

The construction and use of a transformation file is described by J.R. Williams[5]. Briefly the process is as follows. Several locations are identified on the image and the corresponding National Grid coordinates are obtained from maps. The program IM.MATRIX is then used to convert the relative location data into a transformation file containing six numerical values, T1 and T2 referring to the translation of the coordinates and A11 to A22 referring to the shear and rotation of the scene.

The program MAP.TRANS uses a transformation file to convert a map file in image units (IPRJC1 = 99) to another map file in British National Grid coordinates (IPRJC1 = 1) . Each output coordinate-pair XO , YO is calculated from the input coordinates XI , YI :-

$$XO = (XI - T1) \times AI11 + (YI - T2) \times AI12$$
$$YO = (XI - T1) \times AI21 + (YI - T2) \times AI22$$

where AI11 to AI22 is the inverse matrix:-

$$AI11 = A22/D$$
$$AI12 = -A12/D$$
$$AI21 = -A21/D$$
$$AI22 = A11/D$$

and $$D = (A11 \times A22 - A21 \times A12) .$$

The minus signs for AI12 and AI21 are needed because an image file has its Y direction downwards, *ie* in the opposite direction to a map, or map file.

037

It should be noted that the transformation matrix must have been calculated from control points whose image coordinates refer to the same portion of the image which gave rise to the map file. It is not uncommon for a user to select a subscene of an image (using program IM.SUB). A map file made, by means of IM.CONTOUR , from such a subscene could not employ the matrix derived for the whole scene. If this were attempted, the National Grid map would be displaced in northing and easting.

Operation of MAP.TRANS is as follows:

* The program announces its name and date.

* A matrix filename is requested. This file is then opened by the computer, relevant values read and the file closed.

* The input map filename is requested and the file opened. A check is made that the file is in the required pseudo-projection. If it is not, the file is closed and a return is made for a new matrix filename. Assuming a suitable file has been selected, an output filename is requested, and opened; an update is made, including a request for a description of the run; a number of calculations is made and the geographical corner points (the corners expressed as kilometres on the National Grid projection) are printed out, and placed into a header buffer, together with other header data. The output header record is then written to the output medium, and the data of the file is converted to the output projection, and stored. When all data has been transformed, the input and output files are closed.

* MORE FILES? is asked. If YES the program returns to the request for a matrix filename; if NO , it stops.

4 SUBROUTINE LIBRARY

All of the programs described in section 3 call subroutines. Inspection of the program listings reveals that some of the subroutines are called by more than one program. In such cases it is convenient to form a 'library' of subroutines, which are accessible without being re-listed and re-compiled for each program.

The subroutines used by the map file programs and placed into the library include (in alphabetical order):

Subroutine name	Description in section number
NCLOSI	4.2.10
NCLOSO	4.2.6
NFETCH	4.2.9
NHDIN	4.2.8
NOPENI	4.2.7
NOPENO	4.2.1
NSTART	4.2.4
NSTORE	4.2.5
PERMHD	4.2.2
UPDATE	4.2.3 .

Several of these subroutines are concerned with the input and output of data, which is often done by means of routines which are not standard FORTRAN ones. In the subroutine listings, the comment includes references to PDR 3106 and PDR 3110 together with page numbers, on some occasions when non-standard subroutine calls are made. These PDR numbers refer to the host computer manufacturer's handbooks and are detailed in numbers 6 and 7 respectively in the reference list of this document. In passing, it may be noted that if it were necessary to modify the programs for use on a different host computer, many of the alterations would occur in these subroutines at these input and output calls.

4.1 Subroutine groups

The subroutines listed above belong to one or other of two groups; for 'output' or 'input';

(a) subroutines used when a map file is being written or 'output': NOPENO, UPDATE, NSTART, NSTORE, NCLOSO . Additionally, if the map file is being created from a source which is not itself a map file, PERMHD may be used, usually before UPDATE ;

(b) subroutines used when a map file is being read or 'input': NOPENI, NHDIN, NFETCH, NCLOSI .

The subroutines within each group are called in the sequence given above. If both input and output occurs (as for instance in program MAP.TRANS) then calls from the two groups may be intermingled, but the sequence within each group will still be preserved. Programs, such as MAP.HDRED which edit, *ie* change, an existing map file, may present certain problems. Since only one such program exists at present, its special subroutines, for opening and closing files for both input and output, and for writing a header record back into an existing file, are not included in the library.

As far as possible the subroutines contained within group (a) and group (b) have been designed to form self-contained sets, needing the minimum of external information. Subroutine parameters are not used, and the transfer of information is by means of COMMON blocks. For group (a), the output routines, the COMMON blocks are OUTCOM, COORDS, HEADER and FILENC . The input routines use COMMON blocks INCOM, COORDS and HEADER.

The method of use of the various subroutines can be well seen in such programs as MAP.FAB which uses all of group (a) in a simple manner, and MAP.LIST which uses all of the routines of group (b).

4.2 Subroutine descriptions

The subroutines are described in detail in Appendix B, being placed in the sequence indicated by the groupings of section 4.1. These descriptions may advantageously be studied together with the library computer listings at the end of the Report.

The following facts are common to all of the subroutines and are therefore not included in Appendix B for each:

all are written in Prime FORTRAN
all were written by the author of this paper
none has any parameters.

Acknowledgment

The author was for a time with the Experimental Cartography Unit (ECU) of the Natural Environment Research Council, then at South Kensington, London. The map format described in this document bears a considerable resemblance to (but is not exactly the same as) the ECU's 'Lang Format'. Some of the programs described in this Report likewise have some resemblance to programs written by the author whilst at ECU.

Appendix A

EXCHANGE FORMAT

A cartographic data exchange format has been defined by the Experimental Cartography Unit of the Natural Environment Research Council. Since no freely available description exists, this Appendix gives full details of the format.

The purpose of the exchange format is to allow transfer of cartographic data between different establishments, and in particular between different types of computer. The method selected as most suitable to cover the widest range of equipment is to write characters (not numbers) onto magnetic tape.

The format description is:

* Data to be written on magnetic tape: 7 or 9 track: any of the standard densities; parity even or odd; any character set. This allows a wide choice, but the options used must be defined on a separate document.

* The tape may contain one or more sets of data, *ie* 'maps'.

* Each map consists of a number of records, each containing 120 characters.

* Each block written to the magnetic tape must contain the same (integer) number of records. Frequently one record per block is used.

* Each map consists of the sequence:

  ten records containing text
  an unspecified number of data records
  one MK = 6 record (see below) to indicate the end of the map.

* After the final map has been written, including its MK = 6 record, a MK = 30 record is added to indicate the end of the data.

* The text records may contain any message or other information, in character form. Unused portions of records should be filled with space characters or nulls.

* Data records are of four possible types:

  MK = 3 record: start of line
  MK = 4 record: map point
  MK = 5 record: corner point
  record containing line points after the start of the line.

Table A1 lists the contents of the MK records.

Table A1

Contents of MK records

Characters in record	MK = 3	MK = 4	MK = 5	MK = 6	MK = 30
1-2	03	04	05	06	30
3-4	00 (1)	00 (1)	00 (1)	00 (1)	00 (1)
5-10	x	x	x	(2)	(2)
11-16	y	y	y	(2)	(2)
17-18	NFC	NFC	NFC (3)	(1)	(1)
19-25					
⋮	(4)	(4)	(4)	(5)	(5)
115-120					

Notes:

(1) Two zero or space characters.

(2) Six zero or space characters.

(3) For MK = 5 NFC would normally be zero.

(4) Up to 17 six-character feature-codes, followed by space or zero characters. Codes in the range -99999 to 99999.

(5) This portion of the record to be filled with zero or space characters.

(6) For all integers, leading zeros may be replaced by spaces.

(7) All numbers are integers.

(8) x and y coordinates may be expressed in any units but should preferably be defined in an accompanying document.

* If MK = 5 records are supplied (corner points) there must be four of them, representing respectively the SW, NW, NE and SE corners of the map, and they must appear before all other data records, but after the ten lines of text.

* For lines, after the first point has been defined by a MK = 3 record, the remaining points are stored as changes of x and y from the preceding value. The change must never exceed the range -49 to +49 units, and should a larger change occur, extra points must be interpolated. Each change then has the value 50 added to it, to bring it into the range 1 to 99, the value zero being reserved to indicate that the end of the line has been reached.

037

Line point records consist of 30 sets of four characters, two numeric characters for the x change and two for the y change. After the last point of the line, there must be a set of four zeros to indicate the end of the line, even if this necessitates a new record. The remaining unused portion of a record may contain any character.

Appendix B
SUBROUTINE SPECIFICATIONS

B.1 Subroutine NOPENO

Summary - NOPENO is used to open a new map file for output.

Use of COMMON - uses /OUTCOM/. Output arguments:
IOTDEV output device line or unit number
IOTSZ size of output buffer in integer words.

Local storage used - 21 integers; 1 logical variable; 11 bytes for characters; Prime A$KEYS list.

Subordinate subprograms - Prime Applications Library routine OPVP$A

Operation - the COMMON values IOTDEV = 6 and IOTSZ = 1024 are provided for use by following subroutines. The applications library OPVP$A is called, operating in the mode A$OVAP , the effect of which is as follows. The user is asked for a filename which may be a Prime 'tree name'. OPVP$A attempts to open a new file of this name, for output. If the opening operation is not successful, for example an illegal tree name is provided, a fresh name is requested, and this is repeated until a successful operation is achieved. If the name of an existing file is provided, the user is asked if he wishes to alter it. If NO , a new name is requested, but if YES the user is asked whether he wishes to overwrite the existing file or to extend it. The correct usage for map files is either to choose a new filename or else to overwrite an existing but unwanted file.

B.2 Subroutine PERMHD

Summary - the subroutine PERMHD clears the header buffer, obtains the permanent header details from the user and places them in the header buffer.

Use of COMMON - uses /HEADER/ and /OUTCOM/
Input argument IOTSZ in /OUTCOM/ is used.
All words in /HEADER/ have values output.

Local storage used - 1 integer, 30 characters in FORMAT statements.

Subordinate subroutines - none

Operation - The value IOTSZ is used to determine the number of words of the COMMON block /HEADER/ , and these are all cleared to zero. The user is then asked to supply the permanent header details, which consist of the project name, IPROJ , up to 12 characters and the project description, IDESCR , up to 32 characters.

PERMHD is usually only called when a map file is being created. Subsequent alterations to the map file do not necessitate a call of PERMD .

B.3 Subroutine UPDATE

Summary - UPDATE is used to maintain the file log in the header buffer.

Use of COMMON - uses /FILENC/ and /HEADER/
Input arguments: /FILENC/
NPROG(4) name of the program calling UPDATE
NVERS version number of that program
Output arguments: modifications to /HEADER/ words 201 to 800.

Local storage used - 17 integers, 25 characters in FORMAT statements

Subordinate subprograms - Prime file system subroutine TIMDAT

Operation - Map files contain a log in the header.

Each log entry is 40 words long, and there is space for 15 log entries in header words 201 to 800. Each time UPDATE is called it moves the existing 14 most recent entries (words 201 to 760) into the positions for entries 2 to 15 (words 241 to 800) and the 15th entry, if it existed, is overwritten and hence lost. A new entry is then provided in header words 201 to 240 as follows. The name and version number of the calling program (which must be placed in COMMON /FILENC/ by that program) are moved into header words 201 to 205. The log sequence number, in header word 206 is obtained from the sequence number of the previous log, in word 246, incremented by one. For the first UPDATE call word 246 is zero so the first log number is made 1, as required. The Prime subroutine TIMDAT is then called, and this moves information about the date, time and user-code into the dimension IARRAY, from which it is moved into the appropriate header words. A request is then made to the user for details of the computer run being performed and up to 32 characters can be stored.

B.4 Subroutine NSTART

Summary - NSTART is used to write the header record of a map file, and prepare the output buffer and pointer for data.

Use of COMMON - Uses /HEADER/ and /OUTCOM/
Input arguments - all of /HEADER/ ; IOTDEV and IOTSZ (set by subroutine NOPENO)
Output arguments - IOTBUF and IOTPTR, the output buffer and its pointer

Local storage used - 1 integer, 25 characters in FORMAT statements

Subordinate subprograms - Prime file system subroutine PRWF$$

Operation - The entire contents of the COMMON block /HEADER/ are transferred to the output buffer and written to the output file, from its start, by Prime subroutine PRWF$$. The output file is on device line IOTDEV and the buffer size is IOTSZ . If a system failure occurs whilst the write operation is taking place, an error message 'PRWF$$ FAILURE IN NSTART' is sent to the terminal and the program stopped. All values in the output buffer are then cleared (set to zero) and the output buffer pointer, IOTPTR is set to 1 , ready for data to be placed.

B.5 Subroutine NSTORE

Summary - NSTORE is used to place one data item to the output buffer, and when the latter is full to write it to the map file.

Use of COMMON - Uses /COORDS/ and /OUTCOM/
Input arguments (a) in /COORDS/
MK item type indicator
X,Y coordinate pair
NPTS, NPTZ number of 'points' in the item
NFC number of feature-codes in the item
FCODE(17) up to 17 feature-codes
(b) in /OUTCOM/
IOTDEV output file line number - set by NOPENO
IOTSZ output buffer size - set by NOPENO
Output arguments in /OUTCOM/
IOTPTR output buffer pointer
IOTBUF output buffer

Local storage used - 5 integers, 2 real variables, 38 characters in FORMAT statements, Prime list KEYS.F

Subordinate subprograms - Prime file system subroutine PRWF$$

Operation - Map data is stored in the form of items (see the body of this document for details of items), each of which consists of a MK number in the range 0-6, followed by an appropriate amount of map data. In the case of MK = 0, the MK number is not itself stored, but its presence can be inferred. Items are written to the output buffer IOTBUF , which is written to the output storage medium when it has insufficient space for the next item, or following a MK = 6 .

The detailed operation of NSTORE is as follows:
The variable ISWTCH is set to 1 , to control the normal routing of the sub-routine flow. Only for the case MK = 6 can ISWTCH have any other value. The

037

value of MK is checked, and the illegal values, *ie* other than 0, 1, 3-6, cause the subroutine to output the error message MK = XX IN NSTORE and return. The item size in integers ITEMSZ is calculated, from the value of MK and where necessary NFC . A check is then made of whether sufficient space remains in the buffer to place the item. If insufficient space exists, the buffer must be written to the output storage device, using the Prime file systems routine PRWF$$ to add the data to the end of the existing data. Before the buffer is written, it is checked for any empty words, and if one or more empty spaces remain, the first of these has a fill-word placed into it, appropriate to MK = 7. This value of MK is internal to the subroutines NFETCH and NSTORE and need not be considered by the user of the subroutine - indeed it is illegal external to the routine. After the buffer has been written, all of its values are set to zero, and the pointer IOTPTR is set to 1.

If sufficient space exists in the buffer, the next item is placed into it. MK is not stored as such, because it could be confused with data, but has RMARK (1048576.0) added to it. This value has been chosen as it is large enough to avoid confusion with any possible data value, yet is small enough to allow the MK to be extracted from it. The other words in the item are placed as required by the MK value, and may include some or all of: X Y NPTS NPTZ NFC and up to 17 feature codes. To aid in placing the real words, the buffer is equivalenced to a real array OUTBUF and the real buffer pointer IPTR is used to point to the real word locations.

For the case MK = 6 , which indicates that the end of the map file has been reached, the item consists only of one word, RMARK + MK , and this word must be placed into the output buffer and written at once to the storage device. If there is space in the output buffer, the data can be placed and written, but if the buffer is full, the buffer must first be written, then the MK = 6 data placed and then a second 'write' take place, *ie* a second call of PRWF$$. In the case MK = 6 , ISWTCH is set to 2, and this allows the correct routing after the PRWF$$ call, *ie* a return is not made following a PRWF$$ call, when a MK = 6 has been written.

B.6 Subroutine NCLOSO

Summary	- NCLOSO is used to close an output map file
Use of COMMON	- uses /OUTCOM/ . Input argument: IOTDEV - output file line number
Local storage used	- 1 logical variable, 25 bytes for characters, Prime list A$KEYS

Subordinate subprograms - Prime applications library subroutine CLOS$A

Operation - The Prime subroutine CLOS$A is used to close the output map file on line number IOTDEV . If the CLOS$A operation is unsuccessful the message CLOS$A FAILURE IN NCLOSO is output to the terminal and the program is stopped.

B.7 Subroutine NOPENI

Summary - NOPENI is used to open a map file for input

Use of COMMON - uses /INCOM/ . Output arguments:
INDEV input device line number
INSZ size of input buffer in integer words

Local storage used - 21 integers, 1 logical, 10 bytes for characters, Prime A$KEYS list

Subordinate subprograms - Prime Applications Library routine OPNP$A

Operation - the COMMON values INDEV = 5 and INSZ = 1024 are provided, for use by following input file subroutines. The Prime Applications Library routine OPNP$A is called. This outputs the request for an input filename at the terminal, accepts a Prime tree name, checks it for validity and then attempts to open an existing file of that name. If the opening operation is not successful, the process is repeated until a satisfactory open occurs.

B.8 Subroutine NHDIN

Summary - reads one data block from the input map file, and places it into /HEADER/

Use of COMMON - uses /HEADER/ and /INCOM/
Input arguments: /INCOM/ words:
INDEV input file line number
INSZ size of input buffer
Output arguments: /INCOM/ word:
INPTR - input buffer pointer
All of /HEADER/
Temporary use: /INCOM/
INBUF input buffer

Local storage used - 3 integers, 24 bytes for characters, Prime list KEYS.F

Subordinate subprograms - Prime file system subroutine PRWF$$

Operation - The Prime subroutine PRWF$$ is called to read the next INSZ words from the input map file into the input buffer. INSZ has been set to 1024 by NOPENI , and is equal to one block of data on the Prime disc storage on which map files are held. NHDIN is intended to be called before other subroutines which read data from a map file, and it should therefore read the first 1024 words of the file, which is the map file header. If the PRWF$$ operation is unsucessful (ICODE not zero) or the number of words read IRNW is not equal to INSZ , NHDIN sends the message PRWF$$ FAILURE IN NHDIN to the terminal, and then stops program operation.

Following a successful PRWF$$ read operation, the data in the input buffer is transferred into COMMON /HEADER/ . The input buffer pointer INPTR is set to INSZ + 1 to ensure that the following NFETCH call will be forced to start with a read.

B.9 Subroutine NFETCH

Summary - NFETCH is used to retrieve one map 'item' from the input buffer, reading more data from the input file to the buffer when the latter needs replenishment.

Use of COMMON - Uses /COORDS/ and /INCOM/
Input arguments, in /INCOM/:
INDEV input file line number
INSZ size of input buffer
INPTR input buffer pointer
Output arguments:
Some or all of /COORDS/

Local storage used - 4 integers, 3 real words, 37 bytes for characters
Prime list KEYS.F

Subordinate subprograms - Prime File System subroutine PRWF$$

Operation - Map data is stored in the form of 'items', the details of which are controlled by a MK word. The subroutine NFETCH , when called, retrieves the next item from the input map file, and places it into the relevant words of the COMMON block /COORDS/ . Items are read from the map file in blocks of 1024 words, each block containing many items.

037

NFETCH starts by setting all values of /COORDS/ to zero, so that only newly retrieved values are available, not values left over from a previous call. If the input pointer has a value less than the buffer size, there must still be

data remaining within the buffer, so this data is examined. The next data word is studied to see if it contains a valid MK word, and if so, the action appropriate to that MK word is taken. If, however, the next word does not correspond to a valid MK word, the data must be an X coordinate, implying a MK = 0. A MK =7 indicates tha there is no more data in the buffer, and the subroutine path is then the same as for INPTR greater than INSZ, *ie* a new buffer of data is read from the data file by means of a PRWF$$ call. The PRWF$$ call is followed by a check for successful operation, and if unsuccessful the message PRWF$$ FAILURE IN NFETCH is sent to the terminal and the program stopped. Following a successful PRWF$$ call, the input buffer contains new data items.

If the existing data buffer contains more data or if a new buffer of data has been read from the input file, then the next item can be retrieved, the MK number being calculated by subtraction of RMARK, or the value MK = 0 obtained as explained above. The MK value indicates which words are available and these are extracted from the input buffer, and placed into the /COORDS/ words: X, Y, NPTS, NPTZ, NFC and the feature codes FCODE(17). The input buffer pointer INPTR and the real-word buffer pointer IPTR are used to aid this process, pointing to the relevant locations in the equivalent integer or real buffers INBUF and RINBUF. A detailed description of the various types of item is given elsewhere in this document.

If the illegal MK = 2 is found, the message MK = 2 IN NFETCH is sent to the terminal, and a return made.

B.10 Subroutine NCLOSI

Summary - NCLOSI is used to close an input map file

Use of COMMON - Uses /INCOM/. Input argument:
INDEV input file line number

Local storage used - 1 logical variable, 24 bytes for characters,
Prime list A$KEYS

Subordinate subprograms - Prime Applications Library subroutine CLOS$A

Operation - The Prime subroutine CLOS$A is used to close the input map file on line number INDEV. If the CLOS$A operation is unsuccessful the message CLOS$A FAILURE IN NCLOSI is output to the terminal and the program is stopped.

Table 1

HEADER RECORD ALLOCATION

Word No.	Name	FORTRAN format or type	Meaning
1-6	IPROJ	6A2	General name of the project
7-22	IDESCR	16A2	Description of the project
23-40			Spare
41	IPRJC1	Integer	Map projection code
42	IPRJC2	Integer	Map projection subcode
43-44	SCALE	Real	Map scale
45-46	XSCALE	Real	X and Y scale adjustments
47-48	YSCALE	Real	
49-50	SCLLNG	Real	Longitude and latitude (degrees) for specified purposes
51-52	SCLLAT	Real	
53-54	RADEA	Real	Earth's major semi-axis (metres)
55-56	RADEB	Real	Earth's minor semi-axis (metres)
57-80			Spare
81	NLIN	Integer	Number of lines in the file
82	NPT	Integer	Number of points in the file
83	NSTEPS	Integer	Number of line points in the file
84	NSTEPZ	Integer	
85	MNFC	Integer	Maximum number of codes per item
86	NDIFFC	Integer	Number of different codes in the file
87-100			Spare
101-116	CNR	8 real	Corners of the map (mm)
117-132	GCNR	8 real	Corresponding corners, on the Earth
133-200			Spare
201-800	-	See below	15 sets of 40 words, to 'UPDATE'
		5 integers	Program name and version
		1 integer	Data file version number
		3 integers	Date
		1 integer	Time
		3A2	User's code name
		16A2	Details of the current process
		11 integers	Spare
801-1024			Spare

Table 4

RECOGNISED WORDS FOR PROGRAM MAP.HDRED

Keyed word	Effect
END	The data file is closed
(Carriage return)	
HELP	The list of keyed words is displayed on the terminal
IPROJ	Controls header word(s) of these names
IDESCR	
IPRJC1	
IPRJC2	
SCALE	
XSCALE	
YSCALE	
SCLLNG	
SCLLAT	
RADEA	
RADEB	
NLIN	
NPT	
NSTEPS	
NSTEPZ	
MNFC	
NDIFFC	
CNR(A,B)	Controls header dimension CNR(A,B)
GCN(A,B)	Controls header dimension GCNR(A,B) A = 1 or 2 B = 1 to 4 allowed.

REFERENCES

No.	Author	Title, etc
1	NASA	Landsat Data Users' Handbook, Appendix C (1975)
2	A.H. Benny	Coastal definition using Landsat data and potential bathymetric applications. Sixth Annual Conference of the Remote Sensing Society (1979)
3	A.H. Benny	RAE Technical Report in preparation
4	A.P. Colvocoresses	Space Oblique Mercator. Photogrammetric Engineering, 921-926 (1974)
5	J.M. Williams	Geometric correction of satellite imagery. RAE Technical Report 79121 (1979)
6	Prime Computer Inc.	Preliminary documentation release PDR 3106
7	Prime Computer Inc.	Preliminary documentation release PDR 3110

REPORTS QUOTED ARE NOT NECESSARILY
AVAILABLE TO THE PUBLIC

PROGRAMS

```
C PROGRAM MAP.FAB



C PROGRAM MAP.FAB
C TO CREATE A MAP FILE BY KEYING IN FROM A TERMINAL
      DIMENSION ICO(42)
      LOGICAL Q
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/FILENC/NPROG(4),NVERS
      COMMON/HEADER/IHDUM(1024)
      COMMON/OUTCOM/IOTDUM(1027)
$INSERT SYSCOM>A$KEYS
      EQUIVALENCE (MK,ICO(1))
      DATA NPROG,NVERS/8HMAP.FAB ,1/
C
C
C
      WRITE(1,2000)
 2000 FORMAT('MAP.FAB     9-JAN-80')
  100 CALL NOPENO
      CALL PERMHD
      CALL UPDATE
      CALL NSTART
    1 DO 5 J=1,42
    5 ICO(J)=0    /*CLEARS /COORDS/
      WRITE(1,1000)
 1000 FORMAT('MK:')
      READ(1,1001)MK
 1001 FORMAT(I6)
      IF (MK.GT.6 .OR. MK.LT.0 .OR. MK.EQ.2) GOTO 990
      IF (MK.EQ.6) GOTO 800
      WRITE(1,1002)
 1002 FORMAT('X:')
      READ(1,1003)X
 1003 FORMAT(F10.2)
      WRITE(1,1004)
 1004 FORMAT('Y:')
      READ(1,1003)Y
      IF (MK.EQ.0.OR.MK.EQ.1.OR.MK.EQ.5) GOTO 800
    0 WRITE(1,1005)
 1005 FORMAT('NFC:')
      READ(1,1001)NFC
      IF (NFC.EQ.0) GOTO 20
      DO 10 J=1,NFC
      WRITE(1,1008)
 1008 FORMAT('FC:')
   10 READ(1,1003)FCODE(J)
   20 IF (MK.EQ.4) GOTO 800
      WRITE(1,1006)
 1006 FORMAT('NPTS:')
      READ (1,1001)NPTS
      WRITE(1,1007)
 1007 FORMAT('NPTZ:')
      READ(1,1001)NPTZ
  800 CALL NSTORE
      IF (MK.NE.6) GOTO 1
      CALL NCLOSO
      Q= SNO$A('MORE FILES',10,A$NDEF)
      IF (Q) GOTO 100
      STOP ;123456
C
  990 WRITE(1,1050)
 1050 FORMAT('ILLEGAL MK')
```

```
C PROGRAM MAP.FA8


      GOTO 5
      END

C PROGRAM MAP.HDRED


C PROGRAM MAP HDRED
C TO EDIT THE HEADER RECORD OF A MAP FILE
      LOGICAL Q,CHANGE
      DOUBLE PRECISION DITEMS(36),DWORD
      DIMENSION HDR(70),IPTR(33)
      COMMON/FILENC/NPROG(4),NVERS
      COMMON/HEADER/IHDR(40),IPRJC1,IPRJC2,SCALE,XSCALE,YSCALE,
     * SCLLNG,SCLLAT,RADEA,RADEB,IHDUMB(24),
     * NLIN,NPT,NSTEPS,NSTEPZ,MNFC,NDIFFC,IHDUMC(14),
     * CNR(2,4),GCNR(2,4),IHDUMD(68),IHDUME(824)
      COMMON/INCOM/INDUM(1027)
$INSERT SYSCOM>A$KEYS
      EQUIVALENCE (IHDR(1),HDR(1))
      DATA DITEMS /8HIPRJC1  ,8HIPRJC2  ,8HNLIN    ,8HNPT     ,
     * 8HNSTEPS  ,8HNSTEPZ  ,8HMNFC    ,8HNDIFFC  ,8HSCALE   ,
     * 8HXSCALE  ,8HYSCALE  ,8HSCLLNG  ,8HSCLLAT  ,8HRADEA   ,
     * 8HRADEB   ,8HCNR(1,1),8HCNR(2,1),8HCNR(1,2),8HCNR(2,2),
     * 8HCNR(1,3),8HCNR(2,3),8HCNR(1,4),8HCNR(2,4),8HGCN(1,1),
     * 8HGCN(2,1),8HGCN(1,2),8HGCN(2,2),8HGCN(1,3),8HGCN(2,3),
     * 8HGCN(1,4),8HGCN(2,4),8HIPROJ   ,8HIDESCR  ,8HHELP    ,
     * 8HEND     ,8H        /
      DATA IPTR/41,42,81,82,83,84,85,86,22,23,24,25,26,27,28,
     * 51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,1,7/
      DATA NPROG/8HMP.HDRED/,NVERS/1/
C
C
C
      WRITE(1,1000)
 1000 FORMAT('MAP HDRED    9-JAN-80')
  100 CHANGE=.FALSE.
      CALL NOPEN8  /*OPEN INPUT FILE FOR READ/WRITE
      CALL NHDIN /*READ /HEADER/ FROM FIRST FILE BLOCK
      Q=YSNO$A('HEADER',6,A$NDEF)
      IF ( NOT.Q) GOTO 200
      WRITE(1,1002)(IHDR(J),J=1,22)  /*IPROJ AND IDESCR
 1002 FORMAT('PROJECT: ',6A2,';  DESCRIPTION: ',16A2)
      JA=-39
   80 JA=JA+40
      IF (IHDUME(JA).EQ.0) GOTO 200
      JB=JA+28
      WRITE(1,1003)(IHDUME(J),J=JA,JB)  /*UPDATE LINE
 1003 FORMAT(4A2,I2,I3,1X,A2,':',A2,':'A2,I5,2X,19A2)
      GOTO 80
C
  200 WRITE(1,1006)
 1006 FORMAT('ITEM:')
      READ(1,1102)DWORD
 1102 FORMAT(A8)
      DO 210 J=1,36
      ITNO=J
      IF (DWORD.EQ.DITEMS(J)) GOTO 220
  210 CONTINUE
      WRITE(1,1007)
 1007 FORMAT('ITEM NOT RECOGNISED. TYPE "HELP"')
      GOTO 200
  220 IF (ITNO.EQ.35 .OR. ITNO.EQ.36) GOTO 600  /*END OR <CR>
      IF (ITNO.EQ.34) GOTO 500  /*"HELP"
      ISW=1  /*DITEMS 1 TO 8 - INTEGERS
      IF (ITNO.GT.8) ISW=2  /*DITEMS 9 TO 31 - REALS
      IF (ITNO.GT.31) ISW=3  /*DITEMS 32, 33 - CHARACTERS
```

```
C PROGRAM MAP.HDRED


      NS=IPTR(ITNO)
      GOTO (221,222,223),ISW
  221 WRITE(1,1221)DITEMS(ITNO),IHDR(NS)
 1221 FORMAT(A8,'=',I10)
      GOTO 230
  222 WRITE(1,1222)DITEMS(ITNO),HDR(NS)
 1222 FORMAT(A8,'=',F20.10)
      GOTO 230
  223 IF (ITNO.EQ.32) NE=6  /*END WORD OF IPROJ
      IF (ITNO.EQ.33) NE=22 /*IDESCR
      WRITE(1,1223)DITEMS(ITNO),(IHDR(J),J=NS,NE)
 1223 FORMAT(A8,'=',16A2)
  230 Q=YSNO$A('CHANGE IT',9,A$NDEF)
      IF (.NOT.Q) GOTO 200
      CHANGE=.TRUE.
      WRITE(1,1000)
 1000 FORMAT('NEW VALUE:')
      GOTO (231,232,233),ISW
  231 READ(1,1231)IHDR(NS)
 1231 FORMAT(I10)
      GOTO 200
  232 READ(1,1232)HDR(NS)
 1232 FORMAT(F20.2)
      GOTO 200
  233 READ(1,1233)(IHDR(J),J=NS,NE)
 1233 FORMAT(16A2)
      GOTO 200
C
  500 WRITE(1,1500)DITEMS
 1500 FORMAT('ITEMS AVAILABLE ARE:-'/8(2X,A8))
      GOTO 200
C
  600 IF(.NOT.CHANGE) GOTO 610
      CALL UPDATE  /*ONLY UPDATE IF A CHANGE HAS BEEN MADE
      CALL NHDBAK  /*WRITE HEADER BACK TO FILE, IF CHANGED
  610 CALL NCLOSI
      Q=YSNO$A('MORE FILES',10,A$NDEF)
      IF (Q) GOTO 100
      STOP !123456
      END
C*****************************************************************************
      SUBROUTINE NOPENB  /*OPEN A MAP FILE FOR BINARY INPUT OR OUTPUT
      DIMENSION IFILEI(20)
      LOGICAL Q
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1024)
$INSERT SYSCOM>A$KEYS
C
C
      INDEV=5
      INSZ=1024
      NLEN=40
  100 Q=OPNP$A('INPUT FILE',10,A$RDWR,IFILEI,NLEN,INDEV)
      IF (.NOT.Q) GOTO 100
      RETURN
      END
C*****************************************************************************
      SUBROUTINE NHDBAK  /*WRITES THE FIRST BLOCK INTO /HEADER/
      COMMON/HEADER/IHDR(1024)
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1024)
$INSERT SYSCOM>KEYS.F
```

```
C PROGRAM MAP.HDRED


C
C
C
      DO 100 J=1,INSZ
  100 INBUF(J)=IHDR(J)
      CALL PRWF$$(K$WRIT+K$PREA,INDEV,LOC(INBUF),INSZ,INTL(0),
     * IRNW,ICODE)
      IF (ICODE.EQ.0 .AND. IRNW.EQ.INSZ) RETURN
      WRITE(1,1020)IRNW,ICODE
 1020 FORMAT('PRWF$$ FAILURE IN NHDBAK:',2I8)
      RETURN
      END
```

```
C PROGRAM MAP.HPLOT


C PROGRAM MAP.HPLOT
C TO PLOT A MAP FILE ON A HEWLETT-PACKARD PLOTTER
      LOGICAL Q,QC
      INTEGER*4 DPOS
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/HEADER/IHDUM(40),IPRJC1,IHDUMJ,
     * SCALE,IHDUMA(56),CNR(2,4),IHDUMB(900)
$INSERT SYSCOM>A$KEYS
C
C
C
      WRITE(1,1000)
 1000 FORMAT('MAP.HPLOT     22-JAN-80')
  100 CALL HPINIT  /*INITIALIZE PLOTTER
  110 WRITE(1,1002)
 1002 FORMAT('PEN NUMBER:')
      READ(1,1100)J
 1100 FORMAT(I6)
      IF (J.LT.1 .OR. J.GT.4) GOTO 110
      CALL HPEN(J) /*SELECT PEN J
  116 Q=RNUM$A('CROSS SIZE (I6)',15,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 116
      NCROSS=INTS(DPOS)
      CALL HOPENI  /*OPEN INPUT FILE
      CALL HHDIN  /*INPUT THE HEADER BLOCK
  151 W=CNR(1,3)-CNR(1,1)  /*WIDTH OF FRAME
      H=CNR(2,3)-CNR(2,1)  /*HEIGHT OF FRAME
      IF (H.LE.0 .OR. W.LE.0) GOTO 950
      QC=YSNO$A('SPECIFY CORNERS',15,A$DNO)
      IF (.NOT.QC) GOTO 60
  152 Q=RNUM$A('S.W. X',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 152
      IF (DPOS.LT.0 .OR. DPOS.GT.3000) GOTO 152
      SWX=FLOAT(DPOS)
  153 Q=RNUM$A('S.W. Y',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 153
      IF (DPOS.LT.0 .OR. DPOS.GT.2000) GOTO 153
      SWY=FLOAT(DPOS)
  154 Q=RNUM$A('N.E  X',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 154
      IF (DPOS.LE.IFIX(SWX)  OR. DPOS GT 3000) GOTO 154
      RNEX=FLOAT(DPOS)
  155 Q=RNUM$A('N.E. Y',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 155
      IF (DPOS.LE.IFIX(SWY) .OR. DPOS GT 2000) GOTO 155
      RNEY=FLOAT(DPOS)
      PW=RNEX-SWX
      PH=RNEY-SWY
      GOTO 70
C
   58 IF (SCALE.LT.10 0) GOTO 940
   59 Q=RNUM$A('OUTPUT SCALE (I9)',17,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 59
      IF (DPOS.LT.100  OR. DPOS.GT.100000000) GOTO 59
      XSC=SCALE/FLOAT(DPOS)*0.0  /*PLOTTER UNITS ARE 1/8 MM
      IF (W*XSC.GT 3000.0 .OR  H*XSC GT 2000 0) GOTO 930
      GOTO 79
C
   60 SWX=0 0
      SWY=0.0
```

```
C PROGRAM MAP.HPLOT


      IF (IPRJC1.EQ.99) GOTO 65  /*IMAGE-FILE UNITS
      Q=YSNO$A('SPECIFY SCALE',13,A$NDEF)
      IF (Q) GOTO 58
   65 PW=3000.0
      PH=2000.0
   70 XSC=PW/W  /*X SCALING FACTOR TO FIT PLOTTER
      YSC=PH/H  /*Y SCALING FACTOR
      IF (QC) GOTO 80
      XSC=AMIN0(XSC,YSC)  /*CONVERSION FROM FILE (MM) TO TEKTRONIX UNITS
      IF (SCALE.LT.10.0) GOTO 79
      OSCALE=SCALE*8.0/XSC
      WRITE(1,1004)OSCALE
 1004 FORMAT('SCALE PLOTTED=',F16.2)
   79 YSC=XSC
   80 Q=YSNO$A('FRAME',5,A$NDEF)
      IF (.NOT.Q) GOTO 95
      CALL HPMOVE
      DO 90 J=1,5  /*DRAW THE FRAME
      JA=J
      IF (JA.EQ.5) JA=1
      IX=IFIX(CNR(1,JA)*XSC+0.5+SWX)
      IY=IFIX(CNR(2,JA)*YSC+0.5+SWY)
      CALL HPOSPT(IX,IY)
   90 CONTINUE
   95 ITEM=0    /*COUNTS ITEMS (LINE, POINT) IN A SHEET
  200 CALL NFETCH
      IF (MK.GE.6 .OR. MK.LT.0) GOTO 600
      IF (MK.EQ.3 .OR. MK.EQ.4 .OR. MK.EQ.5) ITEM=ITEM+1
      IXP=IFIX(X*XSC+0.5+SWX)
      IYP=IFIX(Y*YSC+0.5+SWY)
      GOTO (250,250,300,400,400),MK /*DUMMY FOR MK=2
C MK=0 OR MK=1. INTERIOR POINT OF A LINE OR END OF LINE
  250 IF (IXP-IXPO.EQ.0 .AND. IYP-IYPO.EQ.0) GOTO 200 /*SAME POINT
      CALL HPOSPT(IXP,IYP)  /*DRAW TO X,Y
      GOTO 310
C MK=3. START OF LINE
  300 CALL HPMOVE  /*SET VECTOR
      CALL HPOSPT(IXP,IYP)  /*MOVE TO X,Y
  310 IXPO=IXP  /*PREVIOUS POINT
      IYPO=IYP
      GOTO 200
C MK=4 OR 5. POINT OR FIDUCIAL.
  400 CALL PLOTCR(IXP,IYP,NCROSS)  /*PLOT POINT AS A CROSS (MY SUBR)
      GOTO 200
C MK=6. END OF THE MAP
  600 WRITE(1,1003)MK,ITEM
 1003 FORMAT('MK=',I6,' AFTER ITEM',I6)
  650 CALL HPEND  /*PURGE BUFFER, IE DRAW EVERYTHING
      CALL NCLOSI  /*CLOSE INPUT FILE
      Q=YSNO$A('ANY MORE FILES',14,A$NDEF)
      IF (Q) GOTO 100
      STOP :123456
C
  930 WRITE(1,1022)
 1022 FORMAT('MAP TOO LARGE AT THIS SCALE')
      GOTO 59
C
  940 WRITE(1,1020)
 1020 FORMAT('INPUT SCALE NOT SPECIFIED')
      GOTO 151
```

037

```
C PROGRAM MAP.HPLOT


C
  950 WRITE(1,1021)CNR(1,1),CNR(2,1),CNR(1,3),CNR(2,3)
 1021 FORMAT(' FIDUCIALS INCORRECT',2(/2F10.2))
      GOTO 650
      END
C*************************************************************************
      SUBROUTINE PLOTCR(IX,IY,N)  /*PLOTS A POINT AS A CROSS, FOR FIPLOT
C
C
      CALL HPMOVE
      CALL HPOSPT(IX+N,IY)
      CALL HPOSPT(IX-N,IY)
      CALL HPMOVE
      CALL HPOSPT(IX,IY+N)
      CALL HPOSPT(IX,IY-N)
      RETURN
      END
```

```
C PROGRAM MAP.IN
```

```
C PROGRAM MAP.IN
C TO READ AN "EXCHANGE FORMAT" MAGTAPE INTO A MAP FILE
      LOGICAL NOCORN
      DIMENSION IBUF2(200)
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/FILENC/NPROG(4),NVERS
      COMMON/HEADER/IHDR(100),CNR(2,4),IHDUMA(900)
      COMMON/INTRDC/IBP,IBUF(200)
      COMMON/OUTCOM/IOTDUM(1027)
      DATA NPROG/8HMAP.IN  /,NVERS/1/
C
C
C
      UNIT=0.02  /*CONVERTS EXCHANGE UNITS TO MM.
      WRITE(1,1000)
 1000 FORMAT(' MAP.IN      9-JAN-80'/' TAPE UNIT NO:')
      READ(1,1100)MTUNIT
 1100 FORMAT(I6)
      CALL REW(MTUNIT)
      NBLOCK=0  /*COUNTS BLOCKS ON MAGTAPE
      NFILE=0  /*COUNTS FILES FROM THIS MT
C START NEXT OUTPUT FILE
   90 LINE=0     /*COUNTS BLOCKS WITHIN ONE MAP
      ITEM=0     /*COUNTS ITEMS (LINE, POINT) IN A SHEET
      NOCORN=.FALSE.  /*FOR FIDUCIAL CHECK
      NFILE=NFILE+1
  100 IND=60  /*EXPECTED NO OF WORDS (120 CHARS)
      CALL MTREAD(MTUNIT,NBLOCK,IND)  /*INCREMENTS NBLOCK AS WELL
      LINE=LINE+1
      IF (IND.EQ.4) STOP :1000  /*EOT
      IF (IND.EQ.3) GOTO 990    /*EOF
      IF (LINE.GT.10) GOTO 110
      IF (LINE.NE.1 .OR. NBLOCK.EQ.1) GOTO 102
      MK=INTRD(2)
      IF (MK.EQ.30) GOTO 980
      WRITE(1,1006)
 1006 FORMAT(' ANOTHER FILE ON TAPE  TYPE S TO CONTINUE')
      PAUSE  /*TYPE "S" TO CONTINUE
  102 DO 105 J=1,120  /*PREPARE IBUF FOR PRINTING
  105 IBUF2(J)=LS(IBUF(J),8) /*SHIFT INTO CORRECT BYTE
      WRITE(1,1002)(IBUF2(J),J=1,120)  /*60-WORD BLOCK
 1002 FORMAT(120A1)
      IF (LINE.NE.10) GOTO 100
      CALL NOPEND  /*OPEN THE OUTPUT FILE
      IF (NFILE.EQ.1) CALL PERMHD  /*GET PERMANENT HEADER DETAILS
      DO 108 J=41,1024
  108 IHDR(J)=0  /*CLEAR HEADER EXCEPT FOR PERM DETAILS
      CALL UPDATE  /*DETAILS OF THIS RUN
      GOTO 100
  110 ITEM=ITEM+1
      MK=INTRD(2)
      IF (MK.GT.6) GOTO 900
      IF (ITEM.LE.4 .AND. MK.NE.5) GOTO 873
      IF (ITEM.GT.4 .AND. MK.EQ.5) GOTO 880
  106 J=INTRD(2)  /*SHOULD BE ZERO
      IF (J.NE.0) WRITE(1,1003)J
 1003 FORMAT(' ICOR=',I6,' IN ITEM',I6)
      IX=INTRD(6)
      IY=INTRD(6)
      X=FLOAT(IX)*UNIT
```

```
C PROGRAM MAP.IN


      Y=FLOAT(IY)*UNIT
      NFC=INTRD(2)
      IF (NFC.LT.1) GOTO 130
      IF (NFC.GT.17) NFC=17
      DO 126 J=1,NFC
      JA=INTRD(6)
  126 FCODE(J)=FLOAT(JA)
  130 IF (ITEM.LE.4) WRITE(1,1004)ITEM,MK,IX,IY,NFC
 1004 FORMAT(I6,',',I4,2I8,I4)
 1008 FORMAT(I6,',',I4,2F8.0,I4)
      GOTO (900,900,300,400,500,600),MK
C
  300 CALL NSTORE  /*MK=3 ITEM
      MK=0 /*INTERIOR LINE POINT
      NPTS=1  /*FIRST POINT WITH MK=3
      NPTZ=0
      NFC=0
  310 IND=60
      CALL MTREAD(MTUNIT,NBLOCK,IND)
      IF (IND.NE.1) WRITE(1,1005)IND,NBLOCK
 1005 FORMAT(' IND=',I6,' IN BLOCK',I6)
      DO 320 J=1,30
      IDX=INTRD(2)-50
      IF (IDX.EQ.-50) GOTO 340  /*END OF LINE
      IDY=INTRD(2)-50
      IF (IDY.EQ.-50) GOTO 340
      NPTS=NPTS+1
      IF (NPTS.LT.10000) GOTO 316
      NPTS=0
      NPTZ=NPTZ+1
  316 IF (NPTS.EQ.2 .AND. NPTZ.EQ.0) GOTO 318
      X=XO
      Y=YO
      CALL NSTORE  /*PREVIOUS POINT, MK=0
  318 XO=X+FLOAT(IDX)*UNIT
      YO=Y+FLOAT(IDY)*UNIT
  320 CONTINUE
      GOTO 310
  340 IF (NPTS.EQ.1 .AND. NPTZ.EQ.0) GOTO 350
      MK=1  /*END OF LINE
      X=XO
      Y=YO
      CALL NSTORE  /*END POINT OF LINE
      GOTO 100
  350 WRITE(1,1007)ITEM
 1007 FORMAT('ITEM',I6,' IS A ONE-POINT LINE')
      GOTO 100
C
  400 MK=4
      CALL NSTORE
      GOTO 100
C
  500 CNR(1,ITEM)=X
      CNR(2,ITEM)=Y  /*FIDUCIALS
      IF (NFC.NE.0) WRITE(1,1009)ITEM,NFC
 1009 FORMAT('ITEM',I6,' HAS NFC=',I4)
      IF (ITEM.LT.4) GOTO 100
      IF (NOCORN) GOTO 950
      CALL NSTART  /*PLACE /HEADER/ INCL CNRS
      MK=5  /*FIDUCIAL
```

```
C PROGRAM MAP.IN


      NFC=0
      DO 520 J=1,4
      X=CNR(1,J)
      Y=CNR(2,J)
  520 CALL NSTORE
      GOTO 100
C
  600 MK=6
      CALL NSTORE
      CALL NCLOSO
      WRITE(1,1020)MK,ITEM
 1020 FORMAT('MK=',I6,' AFTER ITEM',I6)
      GOTO 90
C
  870 NOCORN=.TRUE.
      GOTO 885
  880 MK=4  /*MK 5 IN BODY OF DATA, CHANGED TO MK 4
  885 WRITE(1,1090)MK,ITEM
 1090 FORMAT(' WARNING. MK=',I3,' IN ITEM',I6)
      GOTO 106
  900 WRITE(1,1090)MK,ITEM
      GOTO 100
C
  950 WRITE(1,1021)
 1021 FORMAT(' FIDUCIALS INCOMPLETE')
      CALL NCLOSO  /*USE NDELET WHEN AVAILABLE
      GOTO 990
  980 WRITE(1,1022)MK,NBLOCK
 1022 FORMAT(' MK=',I3,' IN BLOCK',I6)
  990 CALL REW(MTUNIT)
      STOP :123456
      END
C*************************************************************************
      SUBROUTINE MTREAD(MTUNIT,NBLOCK,IND)
C SUBR TO READ ONE BLOCK FROM MT AND PLACE EACH CHAR INTO A WORD OF IBUF
C IND=1: OK   2: NW WRONG   3: EOF   4: EOT   5: ERR
      LOGICAL EOF,EOT,ERR
      COMMON/INTRDC/IBP,IBUF(1)
C
C
      CALL MTRE(MTUNIT,IBUF,200,EOT,EOF,ERR,NW)  /*MTIOLIB SUBR
      NBLOCK=NBLOCK+1
      IF (.NOT.EOF) GOTO 10
      WRITE(1,1000)NBLOCK
 1000 FORMAT(' EOF IN BLOCK',I6)
      IND=3
      RETURN
   10 IF (.NOT.EOT) GOTO 20
      WRITE(1,1001)NBLOCK
 1001 FORMAT(' EOT IN BLOCK',I6)
      IND=4
      RETURN
   20 IF (.NOT ERR) GOTO 30
      WRITE(1,1002)NBLOCK
 1002 FORMAT(' ERROR IN BLOCK',I6)
      IND=5
      RETURN
   30 IF (NW.EQ.IND) GOTO 40
      WRITE(1,1003)NW,NBLOCK
 1003 FORMAT(I6,' WORDS IN BLOCK',I6)
```

```
C PROGRAM MAP.IN


      IND=2
      RETURN
C
   40 DO 50 J=1,NW  /*UNPACK CHARS TO WORDS, BACKWARDS
      K=NW-J+1
      KA=K+K
      N=IBUF(K)
      N1=RT(N,7) /*RIGHT TRUNCATE.  OR USE: AND(N,127)
      N2=RS(N,8)  /*RIGHT SHIFT 8 PLACES
      N2=RT(N2,7)
      IBUF(KA-1)=N2
      IBUF(KA)=N1
   50 CONTINUE
      IND=1
      IBP=1  /*READY FOR USE
      RETURN
      END
C*************************************************************************
      FUNCTION INTRD(N)
C CONVERTS NEXT N (1 TO 6) CHARS TO AN INTEGER.
      LOGICAL NUMST,NEG,ERROR
      DIMENSION ISTRNG(6)
      COMMON/INTRDC/IBP,IBUF(1)
C
C
      ERROR=.FALSE.
      IF (N.LT.1 .OR. N.GT.6) RETURN
      NUM=0
      NUMST=.FALSE.  /*NUMBER NOT STARTED
      NEG=.FALSE.  /*NO MINUS SIGN SEEN YET
      DO 100 J=1,N
      NEXCH=RT(IBUF(IBP+J-1),7)  /*LAST 7 BITS ONLY
      IF (NEXCH.EQ.32 .OR. NEXCH.EQ.43 .OR. NEXCH.EQ.45) GOTO 50 /*SP + -
      NEXCH=NEXCH-48  /*CONVERT TO NUMBER
      IF (NEXCH.LT.0 .OR. NEXCH.GT.9) GOTO 900
      IF (NUM.GT.3276) GOTO 900  /*OUTPUT MUST NOT EXCEED 32768
      NUM=NUM*10+NEXCH
      GOTO 60
   50 IF (NUMST) GOTO 900
      IF (NEXCH.EQ.43) GOTO 60  /* PLUS SIGN
      IF (NEXCH.NE.45) GOTO 100 /* MINUS SIGN
      NEG=.TRUE.
   60 NUMST=.TRUE.
  100 CONTINUE
      IF (NEG) NUM=-NUM
      INTRD=NUM
  150 IBP=IBP+N
      RETURN
C
  900 DO 910 J=1,N
  910 ISTRNG(J)=LS(IBUF(IBP+J-1),8)  /*MOVE TO LEFT BYTE
      WRITE(1,1000)(ISTRNG(J),J=1,N)
 1000 FORMAT('ILLEGAL INTRD STRING: ',6A1)
      INTRD=0
      ERROR=.TRUE.
      GOTO 150
      END
```

```
C PROGRAM MAP.LIST


C PROGRAM MAP.LIST
C TO LIST A MAP FILE
      LOGICAL Q,INTPTS
      COMMON/HEADER/IHDR(100),CNR(2,4),GCNR(2,4),IHDUMA(68),IHDUMB(824)
      COMMON/INCOM/INDUM(1027)  /*NOT NECESSARY?
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
$INSERT SYSCOM>A$KEYS
C
C
C
      WRITE(1,1000)
 1000 FORMAT('MAP.LIST   9-JAN-80')
   10 CALL NOPENI  /*OPEN INPUT FILE
      CALL NHDIN /*READ /HEADER/ FROM FIRST FILE BLOCK
      Q=YSNO$A('HEADER',6,A$NDEF)
      IF (.NOT.Q) GOTO 90
      WRITE(1,1012)(IHDR(J),J=1,22)
 1012 FORMAT('PROJECT: ',6A2,'  DESCRIPTION: ',16A2)
      JA=-39
   80 JA=JA+40
      IF (IHDUMB(JA).EQ.0) GOTO 82
      JB=JA+28
      WRITE(1,1011)(IHDUMB(J),J=JA,JB)  /*UPDATE LINE
 1011 FORMAT(4A2,I2,I3,1X,A2,':',A2,':'A2,I5,2X,19A2)
      GOTO 80
   82 WRITE(1,1010)(CNR(1,J),CNR(2,J),J=1,4)
 1010 FORMAT('CNRS',8F9.2)
      WRITE(1,1013)(GCNR(1,J),GCNR(2,J),J=1,4)
 1013 FORMAT('GCNRS',8F9.3)
   90 Q=YSNO$A('DATA',4,A$NDEF)
      IF (.NOT.Q) GOTO 900
      INTPTS=YSNO$A('INTERIOR LINE POINTS',20,A$NDEF)
      MK=5
      NLIN=0  /*NO OF LINES IN THE FILE
      NPT=0  /*NO OF POINTS IN THE FILE
  100 MKO=MK
      CALL NFETCH
      GOTO (110,120,125,130,140,150,160),MK+1
  110 IF (MKO.EQ.0 .OR. MKO.EQ.3) GOTO 112
      WRITE(1,1020)MK,MKO
 1020 FORMAT('MK',I2,' AFTER MK',I2)
      PAUSE '111
  112 IF (INTPTS) WRITE(1,1003)MK,X,Y
  116 NPTSA=NPTSA+1
      IF (NPTSA.LE.9999) GOTO 100
      NPTZA=NPTZA+1
      NPTSA=0
      GOTO 100
  120 IF (MKO EQ 0 .OR. MKO EQ 3) GOTO 122
      WRITE(1,1020)MK,MKO
      PAUSE '111
  122 WRITE(1,1003)MK,X,Y,NPTZA,NPTSA
      GOTO 116
  125 WRITE(1,1020)MK,MKO
      PAUSE '111
      GOTO 100
  130 IF (MKO EQ.1  OR  MKO.EQ 4  OR  MKO EQ 5) GOTO 132
      WRITE(1,1020)MK,MKO
      PAUSE '111
  132 NLIN=NLIN+1
```

```
C PROGRAM MAP.LIST


      IF (NFC.LE.0) WRITE(1,1003)MK,X,Y,NPTZ,NPTS,NFC
      IF (NFC.GT.0) WRITE(1,1003)MK,X,Y,NPTZ,NPTS,NFC,(FCODE(J),J=1,NFC)
 1003 FORMAT(I2,2F10.2,I3,I4,I3,17F10.1)
      NPTSA=2  /*THIS POINT AND END POINT (MK=1)
      NPTZA=0
      GOTO 100
  140 IF (MKO.EQ.1 .OR. MKO.EQ.4 .OR. MKO.EQ.5) GOTO 142
      WRITE(1,1020)MK,MKO
      PAUSE :111
  142 IF (NFC.LE.0) WRITE(1,1004)MK,X,Y,NFC
      IF (NFC.GT.0) WRITE(1,1004)MK,X,Y,NFC,(FCODE(J),J=1,NFC)
 1004 FORMAT(I2,2F10.2,7X,I3,17F10.1)
      GOTO 154
  150 IF (MKO.EQ.5) GOTO 152
      WRITE(1,1020)MK,MKO
      PAUSE :111
  152 WRITE(1,1003)MK,X,Y
  154 NPT=NPT+1
      GOTO 100
  160 IF (MKO.EQ.1 .OR. MKO.EQ.4 .OR. MKO.EQ.5) GOTO 162
      WRITE(1,1020)MK,MKO
      PAUSE :111
  162 WRITE(1,1003)MK
      WRITE(1,1005)NPT,NLIN
 1005 FORMAT('FILE CONTAINS',I6,' POINTS.',I6,' LINES ')
  900 CALL NCLOSI
      Q=YSNO$A('MORE FILES',10,A$NDEF)
      IF (Q) GOTO 10
      STOP :123456
      END
```

```
C PROGRAM MAP.OUT


C PROGRAM MAP OUT
C TO OUTPUT A MAP FILE TO A MAGTAPE IN "EXCHANGE FORMAT"
      LOGICAL ERROR,Q
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/ERR/ERROR
      COMMON/HEADER/IHDR(1024)
      COMMON/INCOM/INDUM(1027)
      COMMON/INTRDC/IBP,IBUF(200)
$INSERT SYSCOM>A$KEYS
C
C
C
      UNIT=50.0  /*CONVERT MM TO EXCHANGE UNITS
      WRITE(1,1000)
 1000 FORMAT(' MAP.OUT     9-JAN-80'/' TAPE UNIT NO:')
      READ(1,1100)MTUNIT
 1100 FORMAT(I6)
      CALL REW(MTUNIT)  /*MTIOLIB CALL
      NBLOCK=0  /*COUNTS BLOCKS ON MAGTAPE
  100 CALL NOPENI  /*OPEN INTERNAL FILE
      CALL NHDIN  /*READS FIRST BLOCK
C
      WRITE(1,1001)
 1001 FORMAT('WRITE TEN HEADER LINES:')
      DO 110 J=1,10
      READ(1,1101)(IBUF(I),I=1,120)
 1101 FORMAT(120A1)
      DO 104 K=1,120
  104 IBUF(K)=RS(IBUF(K),8)  /*SHIFT TO RIGHT
      CALL MTWRIT(MTUNIT,NBLOCK)  /*MTIOLIB SUBROUTINE
      IF (ERROR) GOTO 999
  110 CONTINUE
C
  200 DO 201 J=1,120
  201 IBUF(J)=:240  /*FILL IBUF WITH SPACES
      IBP=1
  202 CALL NFETCH  /*GET NEXT ITEM
      MKIND=MK+1
      GOTO (210,210,210,240,240,240,260),MKIND
C
  210 X=X*UNIT+0.5
      IF (X.LE.32767.0 .AND. X.GE.-32768) GOTO 212
      WRITE(1,1003)X
      X=0.0
  212 IX=IFIX(X)
      Y=Y*UNIT+0.5
      IF (Y.LE.32767.0  AND. Y.GE.-32768) GOTO 214
      WRITE(1,1003)Y
      Y=0.0
  214 IY=IFIX(Y)
      IDX=IX-IXO  /*CHANGE SINCE PREVIOUS POINT
      IDY=IY-IYO
      IXO=IX  /*SAVE PRESENT POINT, FOR PREVIOUS
      IYO=IY
      IAX=IABS(IDX)
      IAY=IABS(IDY)
      IF (IAX.LT.50 .AND  IAY.LT.50) GOTO 220
C FOR IDX OR IDY MORE THAN 49
      NUM=MAX0(IAX,IAY)
      NUM=(NUM-1)/49+1  /*DIVIDE INTO NUM PIECES
```

```
C PROGRAM MAP.OUT


      RNUM=FLOAT(NUM)
      DX=FLOAT(IDX)/RNUM
      DY=FLOAT(IDY)/RNUM
      DO 216 J=1,NUM
      X=FLOAT(J)*DX  /*EXACT LOCATION
      IX=IFIX(X+0.5) /*NEAREST INTEGER
      IDX=IX-IXO     /*CHANGE SINCE PREVIOUS POINT
      IXO=IX         /*NEW PREVIOUS POINT
      Y=FLOAT(J)*DY
      IY=IFIX(Y+0.5)
      IDY=IY-IYO
      IYO=IY
      GOTO 224
  216 CONTINUE
  217 IF (MK.EQ.0) GOTO 202
  218 CALL INTWRT(0,4)  /*PACK REST OF BLOCK WITH ZEROS
      IPCTR=IPCTR+1
      IF (IPCTR.LT.30) GOTO 218
      CALL MTWRIT(MTUNIT,NBLOCK)
      IF (ERROR) GOTO 999
      GOTO 200
C
  220 IF (IDX.EQ.0 .AND. IDY.EQ.0 .AND. MK.EQ.0) GOTO 202
      NUM=1
C OUTPUT AN INTERIOR LINE POINT, MK=0 OR 1
  224 IDX=IDX+50
      CALL INTWRT(IDX,2)
      IDY=IDY+50
      CALL INTWRT(IDY,2)
      IPCTR=IPCTR+1
      IF (IPCTR.LT.30) GOTO 228
      CALL MTWRIT(MTUNIT,NBLOCK)
      IF (ERROR) GOTO 999
      IBP=1
      IPCTR=0
  228 IF (NUM.GT.1) GOTO 216
      GOTO 217
C
  240 CALL INTWRT(MK,2)  /*POINT, FIDUCIAL OR START OF LINE  MK=3,4,5
      CALL INTWRT(0,2)  /*"ICOR"
      X=X*UNIT+0.5
      IF (X.LE.32767.0 .AND. X.GE.-32768) GOTO 242
      WRITE(1,1003)X
 1003 FORMAT('COORDINATE TOO LARGE TO OUTPUT',F10.0,' UNITS')
      X=0
  242 IX=IFIX(X)
      CALL INTWRT(IX,6)
      Y=Y*UNIT+0.5
      IF (Y.LE.32767.0  AND. Y.GE.-32768) GOTO 244
      WRITE(1,1003)Y
      Y=0
  244 IY=IFIX(Y)
      IF (MK.NE.3) GOTO 245  /*FOR A LINE
      IXO=IX  /*  SAVE PREVIOUS POINT
      IYO=IY
      IPCTR=0  /*FOR FOLLOWING POINTS
  245 CALL INTWRT(IY,6)
      IF (MK.EQ.5) GOTO 248
      IF (NFC.GT.17) NFC=17
      CALL INTWRT(NFC,2)
```

```
C PROGRAM MAP OUT


      IF (NFC.LT 1) GOTO 248
      DO 246 J=1,NFC
      I=IFIX(FCODE(J)+0 5)
      CALL INTWRT(I,6)
  246 CONTINUE
  248 CALL MTWRIT(MTUNIT,NBLOCK)
      IF (ERROR) GOTO 999
      GOTO 200  /*FOR NEXT ITEM
C
  260 CALL INTWRT(MK,2)  /*END OF FILE
      CALL MTWRIT(MTUNIT,NBLOCK)
      IF (ERROR) GOTO 999
      CALL NCLOSI
      Q=YSNO$A('ANY MORE FILES',14,A$NDEF)
      IF (Q) GOTO 100
      IBP=1
      MK=30
      CALL INTWRT(MK,2)
      CALL MTWRIT(MTUNIT,NBLOCK)
      IF (ERROR) GOTO 999
      CALL MTWF(MTUNIT,ERROR,ERROR)  /*WRITE FILE MARK TO TAPE
      IF (ERROR) GOTO 999
      CALL MTWF(MTUNIT,ERROR,ERROR)  /*TWO NEEDED
      IF (ERROR) GOTO 999
      CALL REW(MTUNIT)
      STOP :123456
C
  999 WRITE(1,1090)
 1090 FORMAT('FATAL MAGTAPE ERROR')
      CALL NCLOSI  /*CLOSE ANY OPEN INPUT FILE
      STOP :77777  /*DONT REWIND MT - IT CAN BE INSPCTED FOR DAMAGE
      END
C*************************************************************************
      SUBROUTINE MTWRIT(MTUNIT,NBLOCK)
C TAKES 120 CHARS IN IBUF, FORMS 60 INTEGERS AND WRITES THEM TO MAGTAPE
      LOGICAL EOF,EOT,ERROR
      COMMON/ERR/ERROR
      COMMON/INTROC/IBP,IBUF(1)
C
C
      DO 10 J=1,60
      JB=J+J
      JA=JB-1
      INT=LS(IBUF(JA),8)+RT(IBUF(JB),8)
      IBUF(J)=INT
   10 CONTINUE  /*ALL 120 CHARS NOW PACKED
C
      ERROR= FALSE.
      CALL MTWR(MTUNIT,IBUF,60,EOT,ERROR)
      NBLOCK=NBLOCK+1
      IF (EOT) GOTO 900
      IF (ERROR) GOTO 920
      RETURN
  900 WRITE(1,1000)NBLOCK
 1000 FORMAT('MAGTAPE EOT AT BLOCK',I6)
      ERROR= TRUE
      RETURN
  920 WRITE(1,1001)NBLOCK
 1001 FORMAT('MAGTAPE ERROR IN BLOCK',I6)
      RETURN
```

```
C PROGRAM MAP.OUT


      END
C*************************************************************************
C VERSION OF INTWRT USED UNTIL NOW, BUT COMMENTED OUT AT 11-JAN-80
C      SUBROUTINE INTWRT(I,N)
CC CONVERTS AN INTEGER TO N (1 TO 6) CHARACTERS IN IBUF
CC NEGATIVE NUMBERS NOT YET AVAILABLE
C      LOGICAL ERROR
C      DIMENSION ISTRNG(6)
C      COMMON/ERR/ERROR
C      COMMON/INTRDC/IBP,IBUF(1)
CC
CC
C      ERROR=.FALSE.
C      IF (N.LT.1 .OR. N.GT.6) GOTO 998
C      IF (I.LT.0) GOTO 998  /*NEGATIVE INTEGERS NOT YET AVAILABLE
C      INT=I
C      DO 100 J=1,N    /*PLACE N CHARS, BACKWARDS, INTO ISTRNG
C      IA=INT/10
C      IR=INT-IA*10  /*REMAINDER
C      INT=IA  /*READY FOR NEXT ITERATION
C      ISTRNG(J)=IR+:260  /*PLACE BITS 5-8 (1011XXXX)
C  100 CONTINUE
C      IF (INT.NE.0) GOTO 998  /*INTEGER TOO LARGE FOR N CHARS
CC
C      DO 150 J=1,N  /*PLACE ISTRING, BACKWARDS, INTO IBUF
C      IBUF(IBP)=ISTRNG(N-J+1)
C  150 IBP=IBP+1
C      RETURN
CC
C  998 WRITE(1,1000)I,N
C 1000 FORMAT('ERROR IN INTWRT(',2I6,')')
C      ERROR=.TRUE.
C      RETURN
C      END
C*************************************************************************
C NEW VERSION, TESTED ON VDU, BUT NOT YET ONTO MAGTAPE
      SUBROUTINE INTWRT(I,N)
C CONVERTS AN INTEGER TO N (1 TO 6) CHARACTERS IN IBUF
C SUITABLE FOR -32767 < N < +32767
      LOGICAL ERROR
      COMMON/ERR/ERROR
      COMMON/INTRDC/IBP,IBUF(1)
C
C

      ERROR=.FALSE.
      IF (N.LT.1 .OR. N.GT.6) GOTO 998
      NA=N  /*NO OF CHARS FOR UNSIGNED I
      INT=I
      IF (I.GE.0) GOTO 50  /*NOT A NEGATIVE NUMBER
      NA=NA-1
      INT=-INT
      IBUF(IBP)=:255  /*MINUS SIGN
   50 IF (NA.LT.1) GOTO 998
      IBP=IBP+N  /*POINT TO NEXT INTWRT
      DO 100 J=1,NA    /*PLACE NA CHARS, BACKWARDS, INTO IBUF
      IA=INT/10
      IR=INT-IA*10  /*REMAINDER
      INT=IA  /*READY FOR NEXT ITERATION
      IBUF(IBP-J)=IR+:260  /*PLACE BITS 5-8 (1011XXXX)
  100 CONTINUE
```

```
C PROGRAM MAP.OUT


      IF (INT.EQ.0) RETURN
C
  998 WRITE(1,1000)I,N
 1000 FORMAT('ERROR IN INTWRT(',2I6,')')
      ERROR=.TRUE.
      RETURN
      END
```

```
C PROGRAM MAP_TEK


C PROGRAM MAP_TEK
C TO DISPLAY A MAP FILE ON THE TEKTRONIX 4014
      LOGICAL Q
      INTEGER*4 DPOS
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/HEADER/IHDUM(100),CNR(2,4),IHDUMA(900)
$INSERT SYSCOM>A$KEYS
C
C
C
      WRITE(1,1000)
 1000 FORMAT('MAP_TEK      1-FEB-80')
      CALL TKINIT
   50 Q=YSNO$A('FLASH THE SCREEN',16,A$NDEF)
      IF (Q) CALL CLRTEK  /*FLASH THE SCREEN
      WRITE(1,1001)
 1001 FORMAT('CROSS SIZE (I6):')
      READ(1,*)NCROSS
C 1100 FORMAT(I6)
      CALL NOPENI  /*OPEN INPUT FILE
      CALL NHDIN  /*INPUT THE HEADER BLOCK
      Q=YSNO$A('SPECIFY CORNERS',15,A$DNO)
      IF (.NOT.Q) GOTO 60
   52 Q=RNUM$A('S.W. X',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 52
      IF (DPOS.LT.-1000 .OR. DPOS.GT.5000) GOTO 52
      SWX=FLOAT(DPOS)
   53 Q=RNUM$A('S.W. Y',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 53
      IF (DPOS.LT.-1000 .OR. DPOS.GT.5000) GOTO 53
      SWY=FLOAT(DPOS)
   54 Q=RNUM$A('N.E. X',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 54
      IF (DPOS.LE.IFIX(SWX) .OR. DPOS.GT.5000) GOTO 54
      RNEX=FLOAT(DPOS)
   55 Q=RNUM$A('N.E. Y',6,A$DEC,DPOS)
      IF (.NOT.Q) GOTO 55
      IF (DPOS.LE.IFIX(SWY) .OR. DPOS.GT.5000) GOTO 55
      RNEY=FLOAT(DPOS)
      PW=RNEX-SWX
      PH=RNEY-SWY
      GOTO 70
   60 SWX=0.0
      SWY=0.0
      PW=4095.0
      PH=3000.0
   70 W=CNR(1,3)-CNR(1,1)  /*WIDTH OF FRAME
      H=CNR(2,3)-CNR(2,1)  /*HEIGHT OF FRAME
      IF (H.LE.0 .OR. W.LE.0) GOTO 950
      XSC=PW/W  /*X SCALING FACTOR TO FIT TEKTRONIX 4014
      YSC=PH/H  /*Y SCALING FACTOR
      IF (Q) GOTO 80
      XSC=AMIN0(XSC,YSC)  /*CONVERSION FROM FILE (MM) TO TEKTRONIX UNITS
      YSC=XSC
   80 Q=YSNO$A('FRAME',5,A$NDEF)
      IF (.NOT.Q) GOTO 100
      DO 90 J=1,5
      JA=J
      IF (JA.EQ.5) JA=1
      IXS=IXE  /*RUBBISH FOR J=1
```

```
C PROGRAM MAP.TEK


      IYS=IYE
      IXE=IFIX(CNR(1,JA)*XSC+0.5+SWX)
      IYE=IFIX(CNR(2,JA)*YSC+0.5+SWY)
      IF (J.EQ.1) GOTO 90
      CALL AMLCOT(29)
      CALL POSPT(IXS,IYS)
      CALL POSPT(IXE,IYE)
   90 CONTINUE
      ITEM=0     /*COUNTS ITEMS (LINE, POINT) IN A SHEET
  100 CALL NFETCH
      IF (MK.GE.6 .OR. MK.LT.0) GOTO 600
      IF (MK.EQ.3 .OR. MK.EQ.4 .OR. MK.EQ.5) ITEM=ITEM+1
      IXP=IFIX(X*XSC+0.5+SWX)
      IYP=IFIX(Y*YSC+0.5+SWY)
      GOTO (200,200,300,400,400),MK /*DUMMY FOR MK=2
C MK=0 OR MK=1. INTERIOR POINT OF A LINE OR END OF LINE
  200 CALL POSPT(IXP,IYP)  /*DRAW TO X,Y
      GOTO 100
C MK=3. START OF LINE
  300 CALL AMLCOT(29)  /*SET VECTOR
      CALL POSPT(IXP,IYP)  /*MOVE TO X,Y
      GOTO 100
C MK=4 OR 5. POINT OR FIDUCIAL.  IGNORE FOR THE PRESENT
  400 CALL PLOTCR(IXP,IYP,NCROSS)  /*PLOT POINT AS A CROSS (MY SUBR)
      GOTO 100
C MK=6. END OF THE MAP
  600 CALL AMLCOT(-1)  /*PURGE BUFFER, IE DRAW EVERYTHING
      WRITE(1,1003)MK,ITEM
 1003 FORMAT('MK=',I6,' AFTER ITEM',I6)
  900 CALL NCLOSI  /*CLOSE INPUT FILE
      Q=YSNO$A('ANY MORE FILES',14,A$NDEF)
      IF (Q) GOTO 50
      STOP :123456
C
  950 WRITE(1,1021)
 1021 FORMAT(' CNRS INCORRECT. FILE CLOSED')
      GOTO 900
      END
C*************************************************************************
      SUBROUTINE PLOTCR(IX,IY,N)  /*PLOTS A POINT AS A CROSS, FOR FIPLOT
C
C
      CALL AMLCOT(29)
      CALL POSPT(IX+N,IY)
      CALL POSPT(IX-N,IY)
      CALL AMLCOT(29)
      CALL POSPT(IX,IY+N)
      CALL POSPT(IX,IY-N)
      RETURN
      END
```

```
C PROGRAM MAP.TRANS


C PROGRAM MAP.TRANS
C CONVERTS A MAP FILE FROM "IMAGE" TO B.N.G. USING A MATRIX
      LOGICAL Q
      INTEGER*4 DNUM
      DIMENSION OCNR(2,4)
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/FILENC/NPROG(4),NVERS
      COMMON/HEADER/IHDR(40),IPRJC1,IHDUM,SCALE,IHDUMA(56),
     * CNR(2,4),GCNR(2,4),IHDUMB(692)
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1024)
$INSERT SYSCOM>A$KEYS
      DATA NPROG,NVERS/8HMP.TRANS,1/
C
C
      WRITE(1,1000)
 1000 FORMAT('MAP.TRANS    14-FEB-80')
  100 CALL MATOPN  /*OPEN MATRIX FILE
      MATDEV=INDEV+4
      READ(MATDEV,1001)(INBUF(J),J=1,40)
      WRITE(1,1001)(INBUF(J),J=1,40)
 1001 FORMAT(40A2)
      READ(MATDEV,1002)T1,A11,A12
 1002 FORMAT(3F20.6)
      READ(MATDEV,1002)T2,A21,A22
      WRITE(1,1003)T1,T2,A11,A12,A21,A22
 1003 FORMAT(6F12.4)
      CALL NCLOSI  /*CLOSE MATRIX FILE
C CALCULATE THE INVERSE OF THE MATRIX
      A=A11*A22 - A21*A12
      AI11=A22/A
      AI12=-A12/A
      AI21=-A21/A
      AI22=A11/A
C
  110 CALL NOPENI  /*OPEN INPUT FILE
      CALL NHDIN  /*READ THE HEADER BLOCK
      IF (IPRJC1.EQ.99) GOTO 120  /*IMAGE UNITS
      WRITE(1,1004)IPRJC1
 1004 FORMAT('UNSUITABLE INPUT MAP PROJECTION:',I6)
      GOTO 920
  120 WRITE(1,1006)
 1006 FORMAT('SUBIMAGE TOP LEFT CORNER')
  122 Q=RNUM$A('COLUMN NUMBER',13,A$DEC,DNUM)
      IF (.NOT.Q) GOTO 122
      T1=T1-FLOAT(DNUM)+1.0
  124 Q=RNUM$A('ROW NUMBER',10,A$DEC,DNUM)
      IF (.NOT.Q) GOTO 124
      T2=T2-FLOAT(DNUM)+1.0
      CALL NOPENO  /*OPEN OUTPUT FILE
      CALL UPDATE  /*UPDATE THE HEADER
C CONVERT THE CORNERS AND FIND MAX/MIN
      XMAX=-1000000.0
      YMAX=XMAX
      XMIN=1000000.0
      YMIN=XMIN
      ROW=CNR(2,2)-CNR(2,1)  /*HEIGHT OF INPUT FILE
      DO 140 J=1,4
      XC=CNR(1,J)-T1
      YC=ROW-CNR(2,J)-T2
      XO=XC*AI11 + YC*AI12
```

```
C PROGRAM MAP.TRANS


      YO=XC*AI21 + YC*AI22
      IF (XO.GT.XMAX) XMAX=XO
      IF (XO.LT.XMIN) XMIN=XO
      IF (YO.GT.YMAX) YMAX=YO
      IF (YO.LT.YMIN) YMIN=YO
  140 CONTINUE
      GCNR(1,1)=XMIN
      GCNR(2,1)=YMIN
      GCNR(1,2)=XMIN
      GCNR(2,2)=YMAX
      GCNR(1,3)=XMAX
      GCNR(2,3)=YMAX
      GCNR(1,4)=XMAX
      GCNR(2,4)=YMIN
C
      WRITE(1,1050)XMIN,YMIN,XMAX,YMAX
 1050 FORMAT('GEOGRAPHICAL CORNERS, KM'/'S W.',2F14.3/'N.E.',2F14.3)
C
      CNR(1,1)=0.0
      CNR(2,1)=0.0
      CNR(1,2)=0.0
      CNR(2,2)=YMAX-YMIN
      CNR(1,3)=XMAX-XMIN
      CNR(2,3)=YMAX-YMIN
      CNR(1,4)=XMAX-XMIN
      CNR(2,4)=0.0
      SCALE=1000000.0  /*KM STORED AS MM
      IPRJC1=1  /*BRIT. NAT. GRID.  ASSUMES MATRIX IS IMAGE TO GRID
      CALL NSTART  /*PLACE OUTPUT HEADER
      MK=5
      NFC=0
      DO 160 J=1,4
      X=CNR(1,J)
      Y=CNR(2,J)
  160 CALL NSTORE  /*FIDUCIALS
      ITEM=1  /*COUNT INPUT FIDUCIALS
  200 CALL NFETCH
      IF (MK.GE.6) GOTO 900
      IF (ITEM.GT.4) GOTO 220
      ITEM=ITEM+1
      IF (MK.EQ.5) GOTO 210  /*GOTO 200 IF 819'S NOT WANTED
      WRITE(1,1005)MK
 1005 FORMAT('ITEM',I3,' HAS MK=',I6)
      GOTO 910
  210 MK=4
      NFC=1
      FCODE(1)=819.0  /*STORE OLD FIDUCIALS AS 819s
  220 XC=X-T1
      YC=ROW-Y-T2
      X=XC*AI11 + YC*AI12 - XMIN
      Y=XC*AI21 + YC*AI22 - YMIN
      CALL NSTORE  /*STORE THE CONVERTED CO-ORD
      GOTO 200
C
  900 CALL NSTORE  /*NSTORE THE MK=6
  910 CALL NCLOSO  /*CLOSE OUTPUT FILE
  920 CALL NCLOSI  /*CLOSE INPUT FILE
C
      Q=YSNO$A('MORE FILES',10,A$NDEF)
      IF (Q) GOTO 100
```

```
C PROGRAM MAP_TRANS


      STOP !123456
      END
C NOTE THAT X,Y CONVERSION IS -
C       XC=X-T1
C       YC=Y-T2
C       X=XC*AI11 + YC*AI12
C       Y=XC*AI21 + YC*AI22
C*****************************************************************************
      SUBROUTINE MATOPN  /*OPEN A MATRIX FILE
      LOGICAL Q
      DIMENSION IFILEM(20)
      COMMON/INCOM/INDEV
$INSERT SYSCOM>A$KEYS
C
C
      INDEV=5
      NLEN=40  /*UP TO 40 CHARS IN FILENAME
  100 Q=OPNP$A('MATRIX FILE',11,A$READ,IFILEM,NLEN,INDEV)
      IF (.NOT.Q) GOTO 100
      RETURN
      END
```

SUBROUTINES

```
     SUBROUTINE NOPENO  /*TO OPEN A MAP FILE FOR BINARY OUTPUT
     LOGICAL Q
     DIMENSION IFILEO(20)
     COMMON/OUTCOM/IOTDEV,IOTPTR,IOTSZ,IOTBUF(1024)
INSERT SYSCOM>A$KEYS


     IOTDEV=6  /*OUTPUT FILE ON LINE 6
     IOTSZ=1024
     NLEN=40
 100 Q=OPVP$A('OUTPUT FILE',11,A$WRIT+A$DANF,IFILEO,NLEN,IOTDEV,
    * A$OVAP,0,0)  /*PDR 3106 P23-18
     IF (.NOT.Q) GOTO 100
     RETURN
     END
*****************************************************************************
```

```
      SUBROUTINE PERMHD  /*TO CLEAR HEADER AND OBTAIN PERMANENT HEADER DETAILS
      COMMON/HEADER/IPROJ(6),IDESCR(16),IHDUM(1002) /*PERM HDR
      COMMON/OUTCOM/IOTDUM(2),IOTSZ


      DO 20 J=1,IOTSZ  /*CLEAR /HEADER/
   20 IPROJ(J)=0
      WRITE(1,1000)
 1000 FORMAT('PROJECT (12):')
      READ (1,1001)(IPROJ(J),J=1,6)
 1001 FORMAT(16A2)
      WRITE(1,1002)
 1002 FORMAT('DESCRIPTION (32):')
      READ (1,1001)(IDESCR(J),J=1,16)
      RETURN
      END
*****************************************************************************


      SUBROUTINE UPDATE  /*TO PLACE DETAILS OF CURRENT RUN INTO HEADER
      DIMENSION IARRAY(16)
      COMMON/FILENC/NPROGV(5) /*PROG NAME (4) AND VERSION
      COMMON/HEADER/IHDR(1024)


      DO 100 J=1,560
  100 IHDR(801-J)=IHDR(761-J) /*MOVE DOWN PREVIOUS UPDATES
      DO 110 J=1,5
  110 IHDR(200+J)=NPROGV(J)  /*TRANSFER PROGNAME AND VERSION
      IHDR(206)=IHDR(246)+1  /*OUTPUT VERS NO
      CALL TIMDAT(IARRAY,16)  /*PDR 3110 P 3-54
      IHDR(207)=IARRAY(2)  /*DAY OF MONTH
      IHDR(208)=IARRAY(1)  /*MONTH NUMBER
      IHDR(209)=IARRAY(3)  /*YEAR NUMBER 19XX
      IHDR(210)=IARRAY(4)  /*MINS SINCE MIDNIGHT
      DO 120 J=1,3
  120 IHDR(210+J)=IARRAY(12+J)  /*USER
      WRITE(1,1000)
 1000 FORMAT('DETAILS OF THIS RUN (32):')
      READ(1,1001)(IHDR(J),J=214,229)
 1001 FORMAT(16A2)
      RETURN
      END
*****************************************************************************
```

```
      SUBROUTINE NSTART  /*STORE HEADER, CLEAR IOTBUF, SET IOTPTR
      COMMON/HEADER/IHDUM(1024)
      COMMON/OUTCOM/IOTDEV,IOTPTR,IOTSZ,IOTBUF(1)
 INSERT SYSCOM>KEYS.F


      DO 100 J=1,IOTSZ
  100 IOTBUF(J)=IHDUM(J)  /*TRANSFER HEADER TO OUTPUT BUFFER
      CALL PRWF$$(K$WRIT,IOTDEV,LOC(IOTBUF),IOTSZ,INTL(0),IRNW,ICODE)
      IF (IRNW.EQ.IOTSZ .AND. ICODE.EQ.0) GOTO 140
      WRITE(1,1000)IRNW,ICODE
 1000 FORMAT('PRWF$$ FAILURE IN NSTART:',2I6)
      STOP :1111
  140 IOTPTR=1
      DO 150 J=1,IOTSZ
  150 IOTBUF(J)=0
      RETURN
      END
*****************************************************************************
```

```
      SUBROUTINE NSTORE  /*TO STORE AN ITEM TO A MAP FILE
      DIMENSION IR(2),OUTBUF(1)
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/OUTCOM/IOTDEV,IOTPTR,IOTSZ,IOTBUF(1)
INSERT SYSCOM>KEYS.F
      EQUIVALENCE (R,IR(1)),(IOTBUF(1),OUTBUF(1))
      DATA RMARK /1048576.0/


      ISWTCH=1  /*USED FOR ROUTING THROUGH PRWF$$
      IF (MK.GT.6 .OR. MK.LT.0) GOTO 900
      GOTO (50,900,30,30,50,60),MK
      ITEMSZ=4  /*ITEMSZ IN INTEGERS.  MK=0
      GOTO 100
   30 ITEMSZ=8+NFC+NFC  /*MK=3 OR 4
      GOTO 100
   50 ITEMSZ=6  /*MK=1 OR 5
      GOTO 100
   60 ITEMSZ=2  /*MK=6
  100 IF (IOTPTR+ITEMSZ.LE.IOTSZ+1) GOTO 200 /*SPACE FOR THIS ITEM
      IF (IOTPTR.GT.IOTSZ) GOTO 120  /*NO SPACE LEFT IN BUFFER
      R=RMARK+7  /*"FILL" INDICATOR
      IPTR=IOTPTR/2+1
      OUTBUF(IPTR)=R
  120 CALL PRWF$$(K$WRIT,IOTDEV,LOC(IOTBUF),IOTSZ,INTL(0),IRNW,ICODE)
      IF (IRNW.EQ.IOTSZ .AND. ICODE.EQ.0) GOTO 140
      WRITE(1,1000)IRNW,ICODE
 1000 FORMAT('PRWF$$ FAILURE IN NSTORE:',2I6)
      STOP :1111
  140 IF (ISWTCH.EQ.2) RETURN
      IOTPTR=1
      DO 150 J=1,IOTSZ
  150 IOTBUF(J)=0

  200 IPTR=IOTPTR/2+1
      IF (MK.EQ.0) GOTO 500
      R=RMARK+FLOAT(MK)
      OUTBUF(IPTR)=R
      IPTR=IPTR+1
      GOTO(500,900,300,300,500,600),MK
  300 IR(1)=NFC+32*NPTZ
      IR(2)=NPTS
      OUTBUF(IPTR)=R
      IPTR=IPTR+1
      IF (NFC.LE.0) GOTO 500
      IF (NFC.GT.17) NFC=17
      DO 320 J=1,NFC
      OUTBUF(IPTR)=FCODE(J)
  320 IPTR=IPTR+1
  500 OUTBUF(IPTR)=X
      IPTR=IPTR+1
      OUTBUF(IPTR)=Y
      IPTR=IPTR+1
      IOTPTR=IPTR+IPTR-1
      RETURN
  600 ISWTCH=2
      GOTO 120

  900 WRITE(1,1001)MK
 1001 FORMAT('MK=',I6,' IN NSTORE')
      RETURN
      END
*****************************************************************************
```

```
      SUBROUTINE NCLOSO  /*TO CLOSE AN OUTPUT MAP FILE
      LOGICAL Q
      COMMON/OUTCOM/IOTDEV,IOTPTR,IOTSZ,IOTBUF(1)
 INSERT SYSCOM>A$KEYS


      Q=CLOS$A(IOTDEV)  /*PDR 3186 P23-19
      IF (Q) RETURN
      WRITE(1,1000)
1000 FORMAT('CLOS$A FAILURE IN NCLOSO ')
      STOP 1111
      END
*****************************************************************************


      SUBROUTINE NOPENI  /*TO OPEN A MAP FILE FOR BINARY INPUT
      DIMENSION IFILEI(20)
      LOGICAL Q
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1024)
INSERT SYSCOM>A$KEYS


      INDEV=5  /*INPUT FILE ON LINE 5
      INSZ=1024  /*INPUT BUFFER SIZE
      NLEN=40 /*MAX 40 CHARACTERS IN FILENAME
 100 Q=OPNP$A('INPUT FILE',10,A$READ,IFILEI,NLEN,INDEV)  /*3186 23-19
      IF (.NOT.Q) GOTO 100
      RETURN
      END
*****************************************************************************
```

```
      SUBROUTINE NHDIN  /*READS A RECORD INTO /HEADER/
      LOGICAL ERROR
      COMMON/HEADER/IHDR(1024)
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1024)
INSERT SYSCOM>KEYS.F


      CALL PRWF$$(K$READ,INDEV,LOC(INBUF),INSZ,INTL(0),IRNW,ICODE)
      IF (ICODE.EQ.0 .AND. IRNW.EQ.INSZ) GOTO 50
      WRITE(1,1020)IRNW,ICODE
1020  FORMAT('PRWF$$ FAILURE IN NHDIN:',2I6)
      STOP !1111

   50 DO 100 J=1,INSZ  /*TRANSFER INTO /HEADER/
  100 IHDR(J)=INBUF(J)
      INPTR=INSZ+1  /*FORCE BUFFER READ AT NFETCH
      RETURN
      END
*****************************************************************************
```

```
      SUBROUTINE NFETCH  /*TO FETCH AN ITEM FROM A MAP FILE
      DIMENSION IR(2),RINBUF(1),ICOORD(42)
      COMMON/COORDS/MK,X,Y,NPTS,NPTZ,NFC,FCODE(17)
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1)
INSERT SYSCOM>KEYS.F
      EQUIVALENCE (R,IR(1)),(INBUF(1),RINBUF(1)),(ICOORD(1),MK)
      DATA RMARK /1048576.0/


      DO 50 J=1,42
   50 ICOORD(J)=0  /*CLEARS ALL /COORDS/
      IF (INPTR.LE.INSZ) GOTO 80
   60 CALL PRUF$$(K$READ,INDEV,LOC(INBUF),INSZ,INTL(0),IRNW,ICODE) /*3110 3-23
      IF (IRNW.EQ.INSZ .AND. ICODE.EQ.0) GOTO 70
      WRITE(1,1000)IRNW,ICODE
 1000 FORMAT('PRUF$$ FAILURE IN NFETCH',2I6)
      STOP !1111
   70 INPTR=1
   80 IPTR=INPTR/2+1
      R=RINBUF(IPTR)
      IPTR=IPTR+1
      RMK=R-RMARK
      IF (RMK.LT.0.9 .OR. RMK.GT.7.1) GOTO 100  /*NOT A MK
      MK=IFIX(RMK+0.5)
      GOTO (150,999,130,130,150,160,60),MK
  100 MK=0
      X=R
      GOTO 155
  130 R=RINBUF(IPTR)
      IPTR=IPTR+1
      NFC=RT(IR(1),5)  /*RIGHT 5 BITS
      NPTZ=RS(IR(1),5)  /*HIGHER PART OF WORD
      NPTS=IR(2)
      IF (NFC.LT.1) GOTO 150
      DO 134 J=1,NFC
      FCODE(J)=RINBUF(IPTR)
  134 IPTR=IPTR+1
  150 X=RINBUF(IPTR)
      IPTR=IPTR+1
  155 Y=RINBUF(IPTR)
      IPTR=IPTR+1
  160 INPTR=IPTR+IPTR-1
      RETURN

  999 WRITE(1,1001)MK
 1001 FORMAT('MK=',I6,' IN NFETCH')
      RETURN
      END
*****************************************************************************



      SUBROUTINE NCLOSI  /*TO CLOSE AN INPUT MAP FILE
      LOGICAL Q
      COMMON/INCOM/INDEV,INPTR,INSZ,INBUF(1)
INSERT SYSCOM>A$KEYS


      Q=CLOS$A(INDEV)  /*PDR 3106 P23-19
      IF (Q) RETURN
      WRITE(1,1000)
 1000 FORMAT('CLOS$A FAILURE IN NCLOSI')
      STOP !1111
      END
*****************************************************************************
```


DAT
ILM